

For technology
in Quality



User Manual

Windows CE 5.00 BSP for the TQMa31 module

BSPa31-WinCE.UM.101

23.03.2009

	Date:	Name:	Designation:		
Prepared:	05.05.2008	Ch. Reichherzer	Project:	BSPa31-WinCE	
Checked:	05.05.2009	Ch. Reichherzer	Document no.:	BSPa31-WinCE.UM.101	
Company:			Release status:	Draft	
			Customer:	TQC	Page 1 of 95
File:	BSPa31-WinCE.UM.101.doc				
(C) TQ-Components GmbH. All information contained in these documents is to be treated confidentially. Distribution of presentation and information to third party requires written consent of TQ- Components GmbH.					

Table of Contents

1	About This Manual	7
1.1	Terms and Conventions	7
1.2	Acronyms and Definitions	7
1.3	Liability Disclaimer	9
1.4	Copyright and Licensing Costs	9
1.5	Registered Trademarks	9
2	Introduction	10
3	Contents of the CD	10
4	Installation	11
4.1	Installation of the Platform Builder Updates	11
4.1.1	Which Updates are already installed	11
4.1.2	How to install missing updates	11
4.2	Installation of the BSP	12
4.3	Removal of the TQMa31 BSP	12
4.4	Installation of the demo workspaces	13
4.5	Contents of the demo workspaces	13
4.5.1	Contents of the TQM_a31_SampleWorkspace	13
4.5.1.1	Additional Windows CE components of the TQM_a31_SampleWorkspace	13
4.5.1.2	Additional Drivers of the TQM_a31_SampleWorkspace	14
4.6	Removal of a demo workspaces	15
5	Creating a new OS Design based on the TQMa31 BSP	16
5.1	Creating a new workspace	16
5.2	How to add drivers and OS components	21
5.3	OS Design configuration	22
5.3.1	Memory configuration	22
5.3.2	Build options	22
5.3.3	Language settings	23
5.3.4	Environment variables	24
5.4	Building the OS Design	26
5.5	Downloading the run-time image via Ethernet	26
5.5.1	Configuring Ethernet Connection for Downloading and Debugging	26
5.5.2	Downloading a run-time image into SDRAM using EBOOT	29
5.5.3	Downloading a run-time image into NOR Flash using EBOOT	29
5.5.4	Downloading a run-time image running from SDRAM into NOR Flash using EBOOT (relocated Image)	30
5.5.5	Running a run-time image from NOR Flash using EBOOT	31
5.5.6	Debugging the download connection	31
6	EBOOT Bootloader	33
6.1.1	Configuration options of the EBOOT bootloader	33
6.1.2	Updating the EBOOT bootloader	35
7	Device drivers	36
7.1	Display driver	36
7.1.1	Direct Draw display driver	36
7.1.1.1	Adding the display driver to your OS design	36
7.1.1.2	Configuration of the direct draw display driver	36
7.1.2	Mobile 3D display driver	37
7.1.2.1	Adding the display driver to your OS design	37
7.1.2.2	Configuration of the mobile 3D display driver	37
7.2	VFP (Vector Floating Point) support	37

7.2.1	Adding VFP support to your OS design	37
7.2.2	VFP configuration.....	37
7.3	Network	38
7.3.1	SMSC9215 Fast Ethernet driver	38
7.3.1.1	Adding the SMSC9215 Network Driver to your OS design	38
7.3.1.2	Network Configuration for the SMSC9215 Fast Ethernet Driver	38
7.3.2	IP network printer	39
7.4	USB	40
7.4.1	USB OTG drivers	40
7.4.1.1	Adding the USB OTG drivers	40
7.4.1.2	USB OTG drivers configuration.....	40
7.4.2	USB high speed driver	40
7.4.2.1	Adding the USB high speed driver	40
7.4.2.2	USB high speed driver configuration.....	40
7.4.3	USB Mouse and Keyboard support.....	41
7.4.4	USB printer support.....	41
7.5	SD Host Controller	41
7.5.1	Adding the SD Host Controller driver to your OS design	41
7.5.2	SD Host Controller driver configuration.....	41
7.6	Audio	41
7.6.1	Adding the Audio driver to your OS design	41
7.6.2	Audio driver configuration.....	41
7.7	LM75 Temperature Sensor	41
7.7.1	Adding the temperature sensor driver to your OS design	42
7.7.2	Temperature sensor driver configuration	42
7.7.3	How to use the temperature sensor driver	42
7.8	Serial ports (UARTs)	42
7.8.1	Adding serial drivers to your OS design	42
7.8.2	Serial driver configuration.....	42
7.9	Touch panel support	43
7.9.1	Adding the touch panel driver.....	43
7.9.2	Touch panel driver configuration	43
7.10	Dynamic Voltage and Frequency Control (DVFC)	43
7.10.1	Adding the DVFC driver	43
7.10.2	DVFC driver configuration	43
7.11	I2C bus driver.....	44
7.11.1	Adding the I2C bus driver.....	44
7.11.2	I2C Bus driver configuration	44
7.12	Battery driver.....	44
7.12.1	Adding the Battery driver.....	44
7.12.2	Battery driver configuration	44
7.13	Fake Battery Charger.....	44
7.13.1	Adding the Fake Battery Charger.....	44
7.13.2	Fake Battery Charger configuration	44
7.14	GPT driver.....	44
7.14.1	Adding the GPT driver.....	44
7.14.2	GPT driver configuration	44
7.15	IPU Backlight.....	45
7.15.1	Adding the IPU Backlight driver.....	45
7.15.2	IPU Backlight driver configuration	45
7.16	PC Card driver	45

7.16.1	Adding the PC Card driver	45
7.16.2	PC Card driver configuration	45
7.17	PWM driver	45
7.17.1	Adding the PWM driver	45
7.17.2	PWM driver configuration	45
7.18	RTC	46
7.18.1	Adding the RTC driver	46
7.18.2	RTC driver configuration	46
7.19	Buzzer	46
7.19.1	Adding the Buzzer driver	46
7.19.2	Buzzer driver configuration	46
7.19.3	Buzzer driver usage	46
7.19.4	Buzzer test program	47
7.20	International keyboard driver	47
7.20.1	Languages included:	47
7.20.2	Adding the international keyboard drivers	47
7.20.3	Configuration of the international keyboard drivers	48
7.20.4	Example	49
7.20.5	Setting the default keyboard layout	49
7.21	Matrix keypad driver	50
7.21.1	Adding the matrix keypad driver to your OS design	50
7.21.2	Matrix keypad drive configuration	50
7.22	MCP2515 CAN driver	51
7.22.1	Adding the MCP2515 CAN driver to your OS design	51
7.22.2	MCP2515 CAN driver configuration and usage	51
7.23	TQ_CanCe Test Application	51
7.23.1	Adding the TQ_CanCe Test Application to your OS design	51
7.23.2	TQ_CanCe Test Application configuration	51
7.24	GPIO driver	51
7.24.1	Adding the GPIO driver to your OS design	51
7.24.2	GPIO driver configuration and usage	51
7.25	TQ_Gpio Test Application	52
7.25.1	Adding the TQ_Gpio Test Application to your OS design	52
7.25.2	TQ_Gpio Test Application configuration	52
7.26	Watchdog driver	53
7.26.1	Adding the Watchdog driver to your OS design	53
7.26.2	Watchdog driver configuration and usage	53
7.27	TQ_Wdg Test Application	54
7.27.1	Adding the TQ_Wdg Test Application to your OS design	54
7.27.2	TQ_Wdg Test Application configuration	55
7.28	RNGA	55
7.28.1	Adding the RNGA driver	55
7.28.2	RNGA configuration	55
7.29	NLED	55
7.29.1	Adding the NLED driver	55
7.29.2	NLED configuration	55
8	Developing Applications	55
8.1	SDK for eVC++	56
8.1.1	Creating an SDK from your OS design	56
8.2	SDK for .NET Compact Framework	57
8.2.1	Creating a SDK from your OS design	58

8.3	Developing with embedded Visual C++	61
8.3.1	Installation	61
8.3.2	Establish a connection between eVC and the CE device	62
8.4	Developing with Visual Studio 2005	64
8.4.1	Installation	64
8.4.2	Establish a connection between VS 2005 and the CE device	65
8.4.2.1	Startup over Active Sync / Transport over Ethernet	65
8.4.2.2	Startup and Transport over Ethernet (debugging without Active Sync)	69
8.5	Application deployment	69
8.5.1	Adding applications and files to your OS design	69
8.5.2	Auto Start-up of Windows CE applications	70
8.6	ActiveSync	72
8.6.1	Required Components	72
8.6.2	Establish a Active Sync connection over serial port	74
8.6.3	Establish a ActiveSync connection over USB	75
9	Additional customizations	75
9.1	Windows CE Components	75
9.1.1	Telnet Server	75
9.1.2	FTP Server	76
9.1.3	Hive Based Registry	77
9.2	Changing des appearance of Windows CE	78
9.2.1	Changing the desktop wallpaper	78
9.2.2	Hiding the taskbar	78
9.2.3	Changing the folder name of storage devices	79
9.2.4	Setting up the time zone to GMT	79
10	Appendix	80
10.1	User interface libraries	80
10.1.1	CAN (TQ_CanCe.dll)	80
10.1.1.1	Introduction	80
10.1.1.2	Function description	80
10.1.1.2.1	Return values	80
10.1.1.2.2	Interface access functions	81
10.1.1.2.2.1	TQcan_OpenBoard	81
10.1.1.2.2.2	TQcan_CloseBoard	81
10.1.1.2.3	Bus configuration functions	82
10.1.1.2.4	TQcan_SetBaudrate	82
10.1.1.2.4.1	TQcan_GetBaudrate	82
10.1.1.2.5	Message acceptance filter	83
10.1.1.2.5.1	TQcan_SetAccMask	83
10.1.1.2.5.2	TQcan_SetAccCode	84
10.1.1.2.5.3	TQcan_GetAccMask	84
10.1.1.2.5.4	TQcan_GetAccCode	84
10.1.1.2.6	Control of bus activity	85
10.1.1.2.6.1	TQcan_Start	85
10.1.1.2.6.2	TQcan_Stop	85
10.1.1.2.6.3	TQcan_CanStatus	85
10.1.1.2.7	Message transmission	86
10.1.1.2.7.1	TQcan_TransmitMsg	86
10.1.1.2.7.2	TQcan_RequestMsg	86
10.1.1.2.7.3	TQcan_Pending	86
10.1.1.2.8	Message reception	87

10.1.1.2.8.1	The TQc_CAN_EVT struct	87
10.1.1.2.8.2	TQcan_ConfigEvents	88
10.1.1.2.8.3	TQcan_EventCount	88
10.1.1.2.8.4	TQcan_ReadEvent	88
10.1.1.2.8.5	TQcan_ResetTimestamp	89
10.1.2	General Purpose Input/Output (TQ_gpio.dll)	90
10.1.2.1	Introduction	90
10.1.2.2	Functions description	90
10.1.2.2.1	Return value	90
10.1.2.2.2	Open and close the driver	91
10.1.2.2.2.1	TQgpio_Open	91
10.1.2.2.2.2	TQgpio_Close	91
10.1.2.2.3	Set and get GPIO I/O	92
10.1.2.2.3.1	TQgpio_SetPortValue	92
10.1.2.2.3.2	TQgpio_GetPortValue	92
10.1.2.2.4	Configure GPIO Inputs as Interrupt	93
10.1.2.2.4.1	TQgpio_ConfigurePortInterrupt	93
10.1.2.2.4.2	TQgpio_AcknowledgeInterrupt	93
10.2	Third Party Software	94
10.2.1	PrinterCE.NETCF SDK for Visual Studio 2005	94
10.2.2	Integration	94
10.2.3	Functional range	95
10.2.4	Supported Printers	95

Revision History

Rev.	Date	Name	Pos.	Modification
015	19.02.2009	Ch. Reichherzer		Initial draft
100	23.03.2009	Ch. Reichherzer		BSP release
101	05.05.2009	Ch. Reichherzer		Changed structure of tree "Contents of the CD"

1 About This Manual

1.1 Terms and Conventions

Symbol/Tag	Description
 Warning	This symbol represents important details or aspects for working with the BSP.
Note	Helpful information for working with the BSP.
Filename.ext	This specification is used to state the complete file name with its corresponding extension.
Instructions / Examples	Examples of application. e.g. • Specifying memory partitions • Processing a script
Reference	Cross-reference to another section, figure or table.

1.2 Acronyms and Definitions

The following terminology and abbreviations are used:

Acronym	Full Form
---------	-----------

Acronym	Full Form
BSP	Board Support Package
IDE	Integrated Development Environment
OS	Operating System
SDB	Software Development Board
H/W	Hardware
OAL	OEM Adaptation Layer
OEM	Original Equipment Manufacturer
SDK	Software Development Kit
COM	Communication Ports
IRQ	Interrupt Request

1.3 Liability Disclaimer

The TQ-Components GmbH does not accept guarantee of any kind about the topics, accuracy, completeness or quality of information made available in this manual as well as its further use. Claims lodged against TQ-Components GmbH, related to damages of material or intellectual nature, arising out of the use or non-use of information contained in this manual, or out of the use of incorrect or incomplete information, would not be entertained so long as there is no evidence of intentional or negligent fault on the part of TQ-Components GmbH.

The TQ-Components GmbH reserves the right to change or supplement the contents of this manual, or parts thereof, without prior intimation to this effect.

1.4 Copyright and Licensing Costs

The drivers and utilities that are used for the components as well as the bootloader are subject to the copyright of the respective manufacturer. Licensing conditions of the respective manufacturer should be observed.

Licensing costs for the operating system and applications are not included and must be calculated and stated separately.

There are no further licensing costs for general drivers or similar items.

Remarks:

Since we are dealing with an embedded system, no operating system licenses from OEM or office/consumer sphere need to be installed.

1.5 Registered Trademarks

TQ strives to respect the copyrights of graphics used in other publications and texts, by using graphics created by themselves or accessing license-free graphics and text.

All brand names and trademarks used in this publication and under circumstance protected by third-parties are, without restriction, subject to conditions of the respective valid trademark law and ownership law of the respective owner or proprietor. Conclusion cannot be drawn on the basis of their mere mention that brand names and trademarks are not protected by third-party rights.

2 Introduction

The TQM Board Support Package for Windows CE 5.00 contains all necessary software components to allow a simple and fast startup of application development with Windows CE 5.00 on your TQMa31 hardware platform.

A PC with Windows XP Professional and Microsoft Platform Builder 5.00 is required.

The TQMa31 BSP was created with all updates installed including the December 2007 Update. Please assure that your Platform Builder installation has all these updates installed.

3 Contents of the CD

```
+---WINCE500
|   +---BSP
|   |   \---TQMa31_SampleWorkspace
|   |       \   256MB_SDRAM
|   |           \---TQMa31_WINCE500.101.msi
|   |
|   |---Doc
|   |   \---BSPa31-WinCE.UM.101.pdf
|   |
|   +---EB00T
|   |   +---ADMATEC_800x480
|   |   |   |---128MB_SDRAM
|   |   |   |   \---EB00T.nb0
|   |   |   \---256MB_SDRAM
|   |   |       \---EB00T.nb0
|   |   +---POWERTIP_320x240
|   |   |   |---128MB_SDRAM
|   |   |   |   \---EB00T.nb0
|   |   |   \---256MB_SDRAM
|   |   |       \---EB00T.nb0
|   |
|   |---PlatformBuilderUpdates
|   |   |---WinCEPB50-051231-Product-Update-Rollup-Armv4I.msi
|   |   |---WinCEPB50-061231-Product-Update-Rollup-Armv4I.msi
|   |   |---WinCEPB50-071231-Product-Update-Rollup-Armv4I.msi
|   |   \---Windows CE 5.0 Help Update.EXE
|   |
|   +---SampleRunTimeImages
|   |   +---TQMa31_SampleWorkspace
|   |       \---256_SDRAM
|   |           \---NK.bin
|   |
|   +---SampleWorkspaces
|   |   \---PBWorkspaces
|   |       \---TQMa31_SampleWorkspace
|   |
|   +---SDKs
|       \---TQMa31_SampleWorkspace
|           \---TQMa31_SampleWorkspace_SDK_v1.0.1.msi
```

4 Installation

The following chapters will guide you through the installation of the TQMa31 BSP for Windows CE 5.00. We assume that you have already installed the Microsoft Platform Builder 5.00 from original Microsoft DVD or CDs. If not please install it before proceeding to the next paragraph.

4.1 Installation of the Platform Builder Updates

Microsoft delivers updates to the Platform Builder monthly. The TQM BSP was created with all updates installed including the December 2007 Update. Please assure that your Platform Builder installation has all these Updates installed.

4.1.1 Which Updates are already installed

You can check for installed updates by looking in the Platform Builder update folder (C:\WINCE500\Updates). For each installed Update there should be file in HTML and RichText format with detailed information about the update.

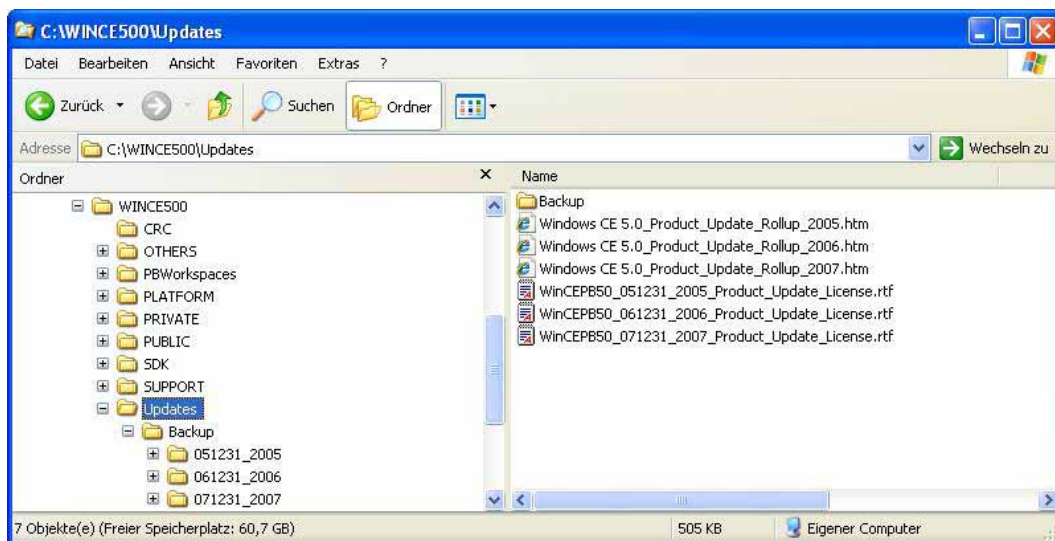


Figure 1: Rollup Updates

4.1.2 How to install missing updates

All Updates released by Microsoft until December 2007 are included on the TQMa31 BSP CD in the Folder PlatformBuilderUpdates. Please install them according to there release date. Start with the Windows WinCEPB50-051231-Product-Update-Rollup-Armv4I.msi update which contains all Updates till December 2005. Next install the WinCEPB50-061231-Product-Update-Rollup-Armv4I.msi and the WinCEPB50-071231-Product-Update-Rollup-Armv4I.msi update.

Additionally the Platform Builder Help Update is included on the CD. Although this update is not needed to use the TQMa31 BSP it will improve the content of the Platform Builder help system.

4.2 Installation of the BSP

The BSP is delivered as an MSI file which can be easily installed by double clicking. The TQMa31 BSP can be found on the CD in the BSP directory.

Note: If you had already installed a older version of this BSP please first remove the old installation (see chapter Removal of the TQMa31 BSP).

First close the Platform Builder IDE if open. Then browse to the BSP folder on the CD and double click the TQMa31_WINCE500.100.msi file. You will be guided through the installation of the BSP.

The setup program will add the BSP to the (\$_WINCEROOT)\PLATFORM directory of your Platform Builder installation.

Additionally it will copy the TQMa31.cec catalog file to the (\$_WINCEROOT)\PUBLIC\COMMON\OAK\CATALOG\CEC directory and add the TQMa31 BSP to your catalog.

Now restart the Platform Builder IDE. In the catalog view you will see the TQM BSP and its device driver components included under (**Catalog -> Third party -> BSPs -> TQMa31**)

4.3 Removal of the TQMa31 BSP

Open the Platform Builder IDE and select *Manage Catalog Items...* from the *File* menu. Now search the TQMa31.cec catalog item and click *Remove*.

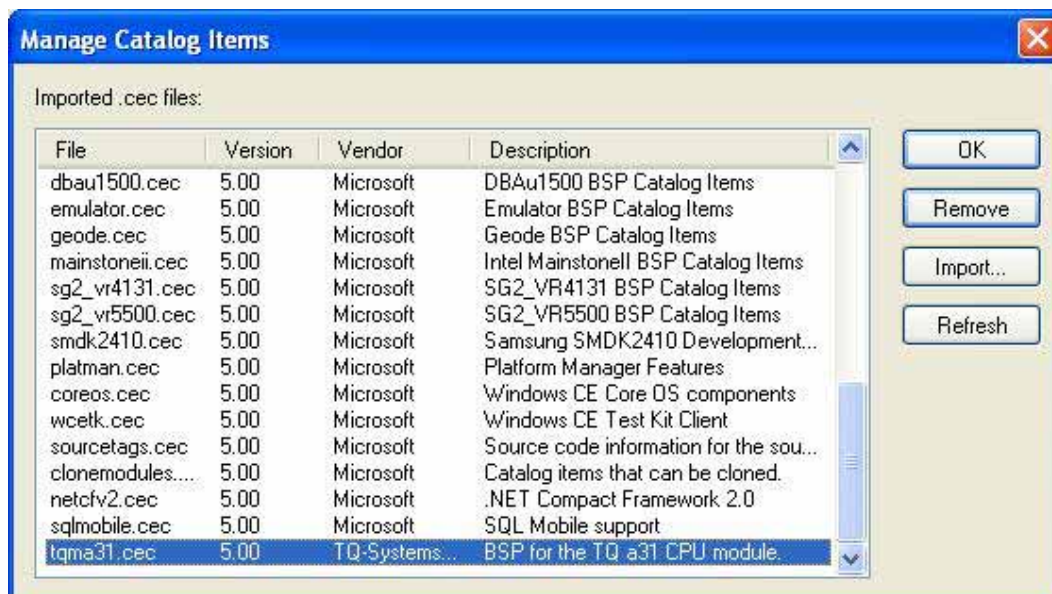


Figure 2: Manage Catalog Items

After closing the dialog box by clicking OK the TQMa31 BSP will disappear from the catalog.

Now you can delete the directory \$(_WINCEROOT)\PLATFORM\TQMa31 and all its subdirectories.

Delete the TQMa31.cec CE Components file from
 (\$ _WINCEROOT)\PUBLIC\COMMON\OAK\CATALOG\CEC.

4.4 **Installation of the demo workspaces**

The demo workspaces for the TQMa31 BSP are included on the CD. The workspaces are a good starting point for a new OS design. They are based on the „Industrial Controller“ design template and have the most common OS design specific drivers and OS components already included.

The workspaces are located in the directory \SampleWorkspaces\PBWorkspaces.

- Copy the appropriate directory for your hardware platform to the PBWorkspaces folder in your Platform Builder installation (e.g. to (\$ _WINCEROOT)\PBWorkspaces). Now you can open it from within the Platform Builder: From Platform Builder *File* menu, select *Open Workspace*.
- Browse to the (\$ _WINCEROOT)\PBWorkspaces\TQMa31_SampleWorkspace folder and open the TQM_a31_SampleWorkspace.pbxml file.

4.5 **Contents of the demo workspaces**

The demo workspaces TQM_a31_SampleWorkspace is based on the „Industrial Controller“ design template.

4.5.1 **Contents of the TQM a31 SampleWorkspace**

The demo workspace TQM_a31_SampleWorkspace includes the following basic components in addition to the „Industrial Controller“ design. For detailed information about the included components please have a look into the workspace file in the folder
 \SampleWorkspaces\PBWorkspaces\TQMa31_SampleWorkspace.

4.5.1.1 **Additional Windows CE components of the TQM a31 SampleWorkspace**

SYSGEN_DOTNETV2	.Net Compact Framework 2.0
SYSGEN_DOTNETV2_SUPPORT	.Net Compact Framework 2.0
SYSGEN_MSXML_DOM	XML Core Services and Document Object Model (DOM)
SYSGEN_CONSOLE	Console Windows
SYSGEN_CONNMC	Network User Interface
SYSGEN_USDK	Standard SDK for Windows CE
SYSGEN_AUDIO	Waveform Audio
SYSGEN_PIE	Pocket Internet Explorer
SYSGEN_IE_JSCRIPT	JScript
SYSGEN_FTPD	FTP Server
SYSGEN_TELNETD	Telnet Server
SYSGEN_HTTPD	Web Server
SYSGEN_DDRAW	Direct Draw

SYSGEN_USB_HID_CLIENTS	USB Hid Keyboard and Mouse
SYSGEN_USB_STORAGE	USB Storage Class Driver
SYSGEN_FATFS	FAT File System
SYSGEN_TFAT	TFAT File System
SYSGEN_SD_MEMORY	SD Memory (SDCard support)
SYSGEN_FSRAMROM	RAM and ROM Filesystem
SYSGEN_FSREGRAM	RAM-based registry
SYSGEN_USBFN_SERIAL	USB Function Client Serial
SYSGEN_STOREMGR_CPL	Storage Manger Control Panel Applet
SYSGEN_PLATMAN	Platform Manager Support

4.5.1.2 Additional Drivers of the TQM a31 SampleWorkspace

BSP_BACKLIGHT	Backlight Driver
BSP_AUDIO_MC13783	Audio Driver
BSP_PMIC_MC13783	PMIC Driver
BSP_BATTERY	Battery Driver
BSP_PMIC_FAKE_CHARGER	Fake Battery Charger
BSP_PWM	PWM Driver
BSP_BUZ	Buzzer Driver
BSP_DISPLAY_ADMATEC_800_480	Powertip Display Driver
BSP_DVFC_MC13783	DVFC Driver
BSP_I2CBUS	I2C Driver
BSP_KBDMOUSE_EVBKPD	International Keybord Driver
BSP_TEMP_LM75	LM75 Temperature Driver
BSP_ETHER_SMSC9215	SMSC9215 Ethernet Driver
BSP_PCMCIA	PCMCIA Driver
BSP_RTC_MC13783	RTC Driver
BSP_SDHC1	SDHC Driver (SDCard)
BSP_GPT	General Purpose Timer
BSP_RNGA	Random Number Generator
BSP_RAM_256	256MB RAM
BSP_MCP2515	CAN Driver
BSP_FSL_VFP	VFP Support
BSP_TOUCH_MC13783	Touch Driver
BSP_TQMA31_WDG	Watchdog Driver

BSP_TQMA31_GPIO	GPIO Driver
BSP_USB_HSOTG_HOST	USB OTG Host Support
BSP_TQ_BuzTest	Buzzer Test
BSP_TQ_CanTest	CAN Test
BSP_TQ_GpioTest	GPIO Test
BSP_TQ_WdgTest	Watchdog Test

4.6 **Removal of a demo workspaces**

Close the Platform Builder IDE.

Delete the appropriate demo workspace directory from the PBWorkspaces directory of your Platform Builder installation (e.g. (\$_WINCEROOT)\PBWorkspaces\TQM_a31_SampleWorkspace).

5 Creating a new OS Design based on the TQMa31 BSP

The following steps will guide you through the whole process of creating, compiling and test of a new Windows CE workspace for the TQMa31 hardware module based on the industrial controller design template.

5.1 Creating a new workspace

Open Platform Builder and select *New Platform...* from the *File* menu. Choose a name for the Workspace. In the next dialog you can specify one or more BSPs on which your OS design should be based on.

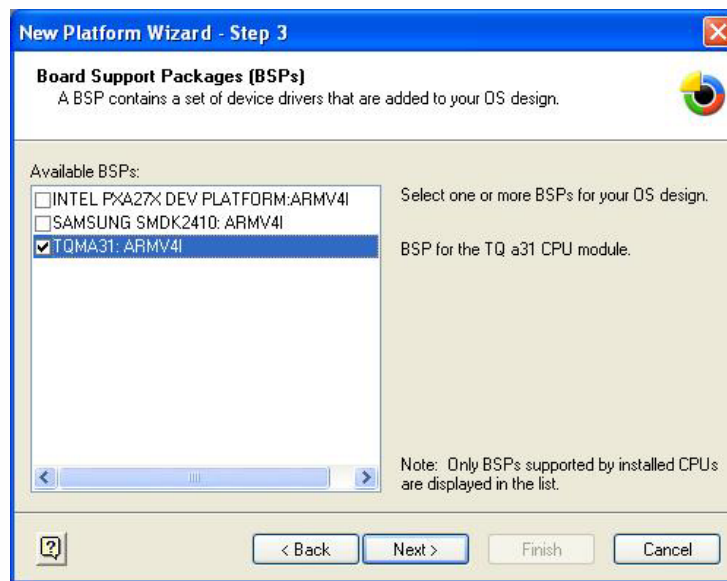


Figure 3: Creating a new platform Step 3

Please choose the TQMa31 BSP as base for your new OS design.

The next step is to choose a predefined design template of the OS design being created. You should select the configuration which has the best conformity to the OS design you need.

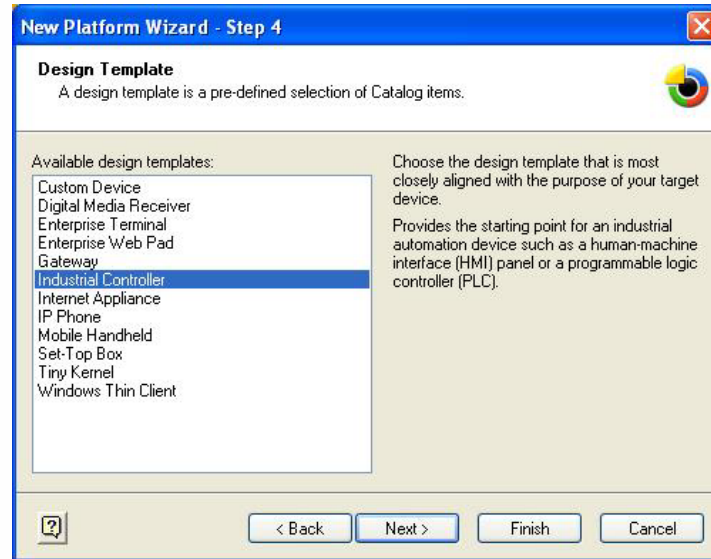


Figure 4: Creating a new platform Step 4

We chose the "Industrial Controller" as a base configuration for our OS design, as this configuration includes already TCP/IP networking, GUI support and most of the programming APIs.

Note: If you want to know exactly what components are included in each design template, please study the .xml files in the `$(_WINCEROOT)\PUBLIC\COMMON\OAK\CATALOG\NEWPLATFORMWIZARDS\` folder of your Platform Builder installation.

With the following two steps various components such as applications can be added to the OS design. You will be able to modify this configuration later on, however it will be more detailed and you will have to deal with dependencies of the modules. The figures below show a configuration example:



Figure 5: Creating a new platform Step 5

We deselect the *.NET Compact Framework* component and include the *Console Window*, the *Standard SDK for Windows CE* and the *Pocket Internet Explorer*.

Console Window: Gives us a command line interface very similar to the cmd.exe in Windows XP.

Standard SDK: Adds the APIs for the Windows CE standard SDK. So applications compiled with the Standard SDK will run on our OS design.

Pocket Internet Explorer: A simple Web Browser.

On the next tab we add the FTP and Telnet server components.

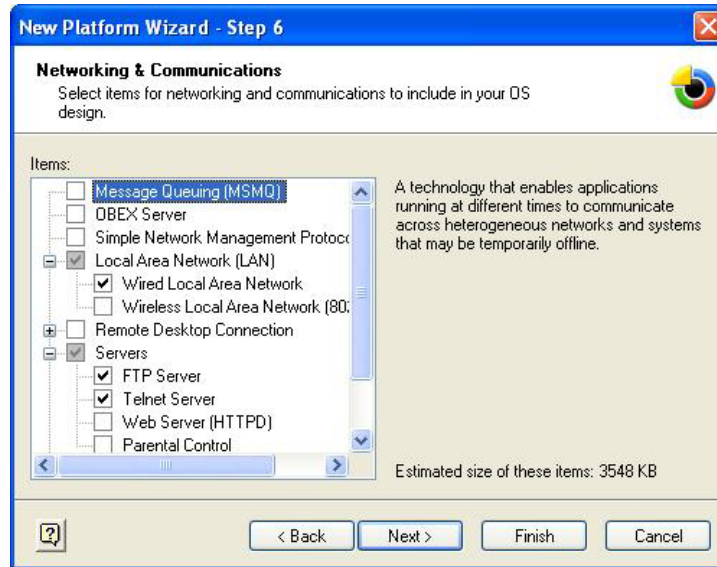


Figure 6: Creating a new platform Step 6

When you click *Next*, a security warning about the FTP and Telnet servers will appear as these components allow network access to our device. FTP and Telnet security settings are discussed later on. Click *Next* and then *Finish*.

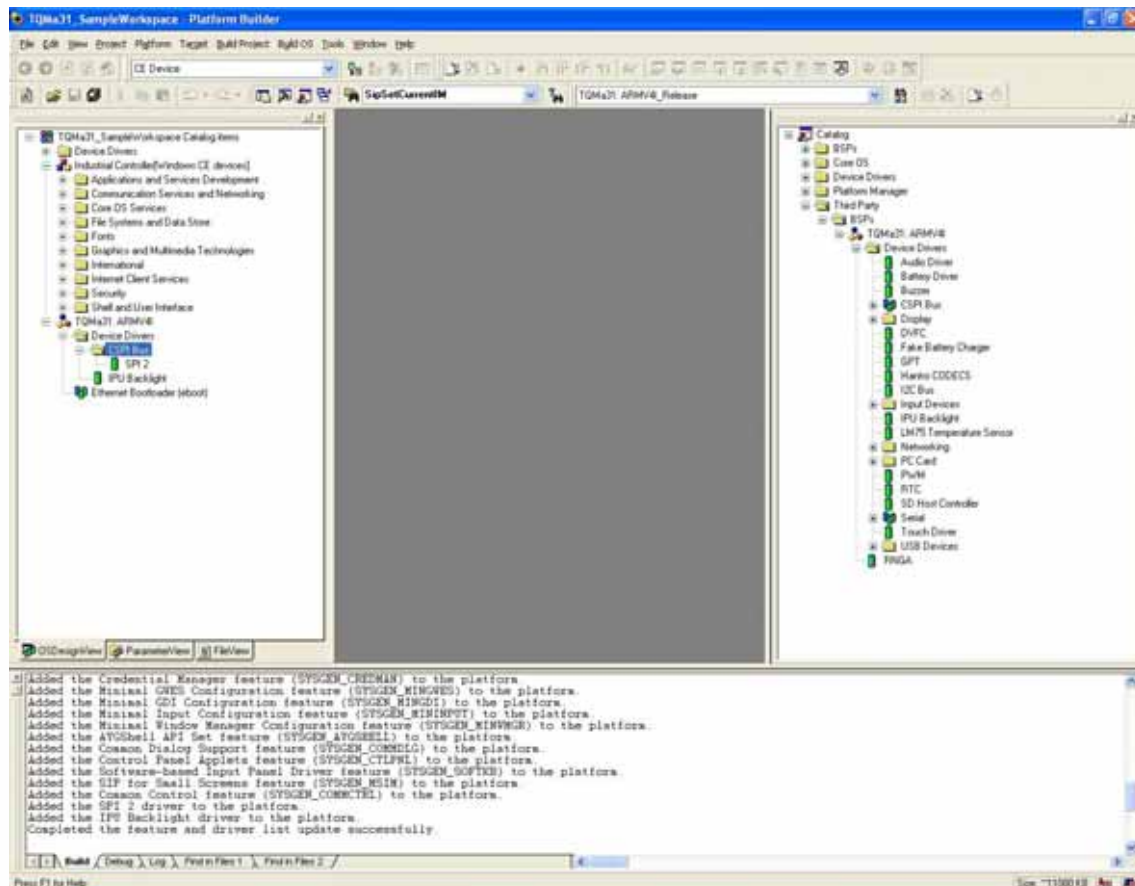


Figure 7: Appearance of platform builder after generating an new platform

Now you have a basic OS design configuration which is ready for building and creating a run-time image, but still lacks the OS design specific hardware drivers.

5.2 How to add drivers and OS components

Drivers can be added to your OS design by right-clicking on the component in the Platform Builders Catalog view and selecting „Add to OS Design“. In the following Example the „SMSC9215 - NDIS“ network driver is added to our workspace.

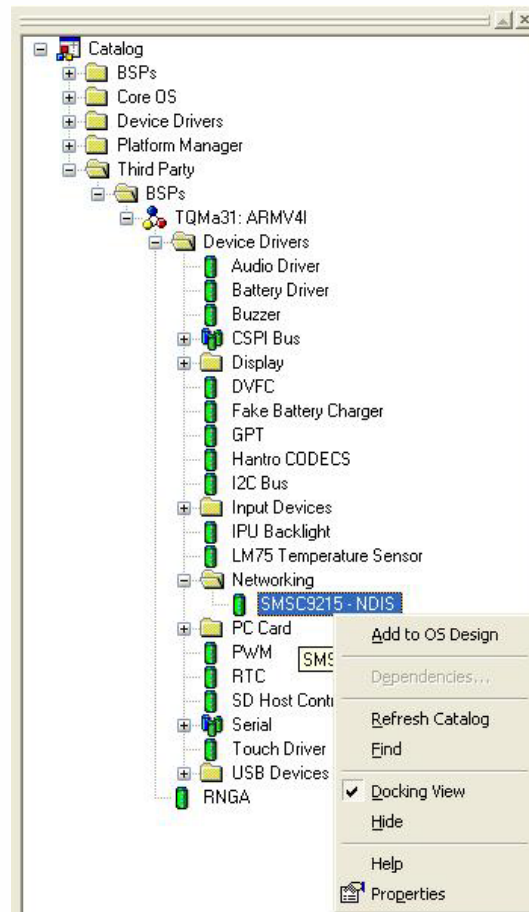


Figure 8: Adding drivers and OS components

The selected component will now be shown in your OS Design View, on the left side of the Platform Builder window.

To remove a driver or component from your OS Design right-click on the component in the Platform Builder's „OS Design“ view and choose „Delete“. The component will be removed from the OS Design Catalog configuration.

5.3 OS Design configuration

After we have added all desired components and drivers, we have to do some configuration settings before we can build our OS design.

5.3.1 Memory configuration

Depending on your TQMa31 module, choose the right memory (SDRAM and NOR Flash) configuration from the catalog.

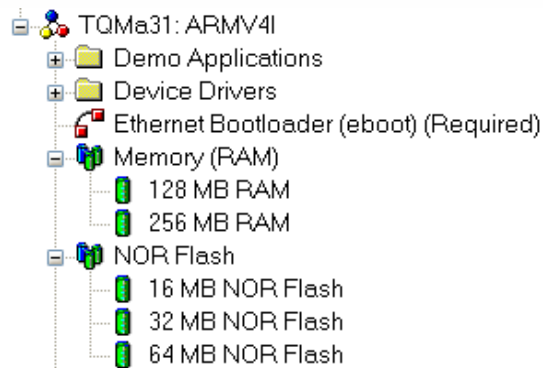


Figure 9: Memory configuration

5.3.2 Build options

To set the build options for your OS design, open the „Settings...” dialog from the *Platform* menu in Platform Builder and go to the „Build Options” tab.

The most important options are described in Table 1. There is also a recommendation given for setting the build options for the Debug and Release configuration of your OS design.

Option	Description	Build configuration	
		Debug	Release
Enable CE Target Control Support	Select to enable target control support (CE Shell) during the boot process.	X	(X)
Enable eBoot Space in Memory	Should be checked if you use the eBoot bootloader (normally used for downloading the Image from the Platform Builder)	X	(X)
Enable Full Kernel Mode	Select to run threads in kernel mode. (Increases system performance, but makes the system more vulnerable)	X	X
Enable Kernel Debugger	Allows to connect the kernel debugger of platform builder with the target.	X	
Enable KITL	Enables the Kernel Independent Transport Layer (KITL). Needed for communication between the development workstation and the target device.	X	(X)

Table 5-1: Build options

In Release Builds the KITL should always be turned off as it will slow down your platform performance!

5.3.3 Language settings

To set the default language of your OS design open the „Settings...“ dialog from the *Platform* menu in Platform Builder and go to the „Locale“ tab.

Here you can set the locales that your OS design will support. The locales include information about currency formats, date and time formats, etc. specific to each country.

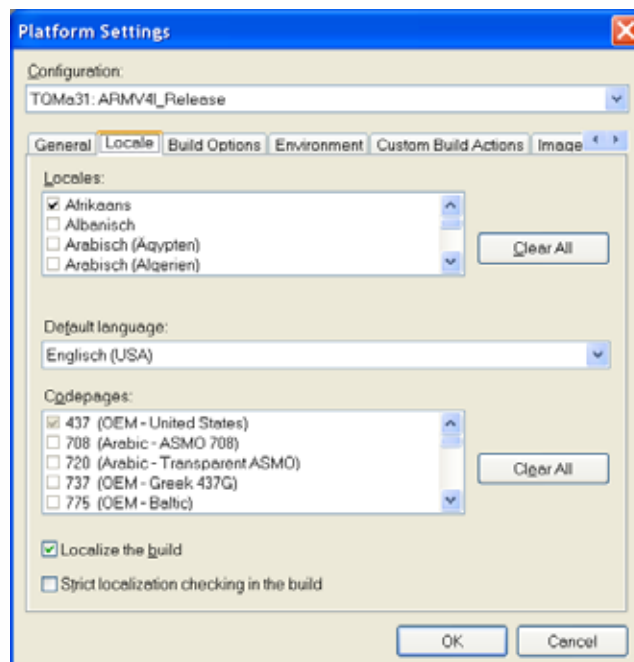


Figure 10: Locale settings

The default language specifies the language of the Windows CE user interface (buttons, menus, windows, etc.).

To set the default keyboard layout, which is independent from the default language, please refer to Chapter [Setting the default keyboard layout](#).

5.3.4 Environment variables

The build process of a OS design is controlled by environment variables. In the tab Build OS -> Open Release Directory you can set additional environment variables for your workspace.

To check which environment variables are set, open the release directory (Build OS -> Open Release Directory):

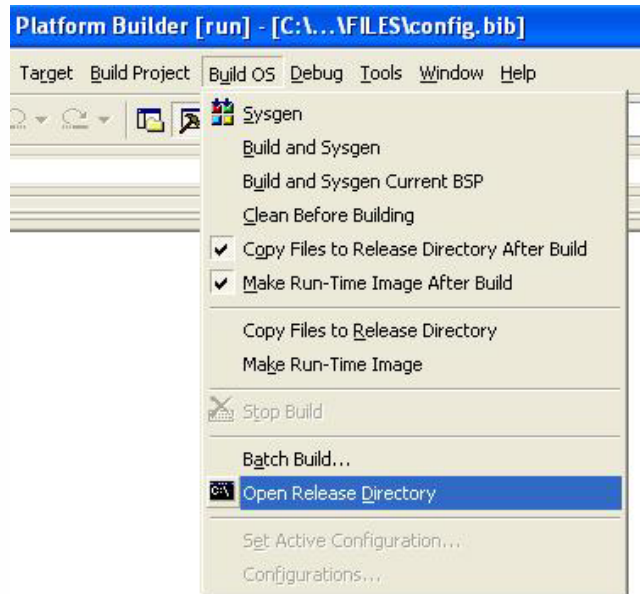


Figure 11: Open Release Directory

Use the set command to get a list of currently set environment variables for your OS design:

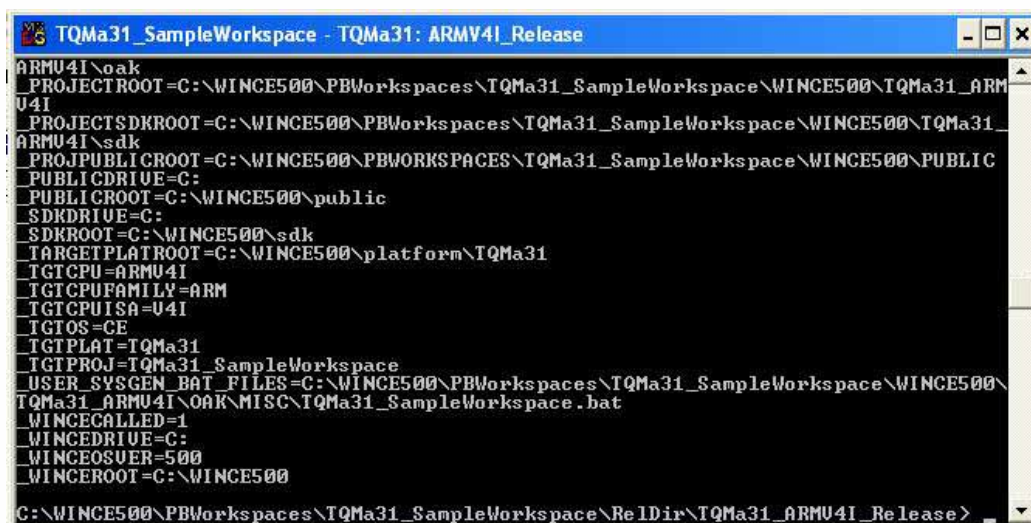


Figure 12: Viewing environment variables

Example:

If you want to set the *environment variable* `BSP_HIVE_CFCARD`, please type:

Set `BSP_HIVE_CFCARD=1`

To delete the variable, just type:

Set `BSP_HIVE_CFCARD=`

Alternatively you can add the environment variable in the Environment tab of the Platform Settings dialog.

5.4 Building the OS Design

To build your OS design, first choose the desired build configuration (Debug or Release) from the pull-down menu in the Platform Builder IDE. You can also select the build configuration by selecting *Set active configuration...* from the *Build OS* menu in Platform Builder IDE.

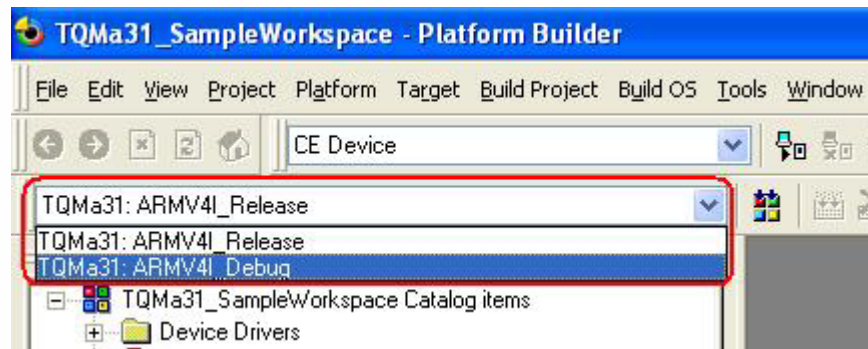


Figure 13: Selecting the active configuration

Second start the system generation: Click the Sysgen button or choose *Sysgen* from the *Build OS* menu.

The build process can take up to 20 minutes to build the run-time image, depending on the number of included components.

After a successful build, the Windows CE run-time image (a file called NK.BIN respectively NK.nb0) can be found in the Release Directory. E.g. for the sample workspace *TQMa31_SampleWorkspace* and the release configuration this would be:

`$(_WINCEROOT)\PBWorkspaces\TQMa31_SampleWorkspace\RelDir\TQMa31_Release\NK.BIN`

5.5 Downloading the run-time image via Ethernet

The Ethernet boot loader (EBOOT) is used to download and execute OS images. EBOOT is typically written into NOR flash memory on the TQMa31 module and executes immediately out of reset. Initially, the target hardware will have EBOOT resident in the NOR flash memory. In addition, the target hardware has non-volatile storage for the EBOOT network configuration (DHCP/static, MAC address, etc.) that must be initialized before using the boot loader with Platform Builder. This section will describe the procedure for updating, and configuring EBOOT on the target hardware.

5.5.1 Configuring Ethernet Connection for Downloading and Debugging

To configure an Ethernet connection that can be used for downloading and debugging images, follow these steps:

1. From the Platform Builder *Target* menu select *Connectivity Options...*
Then select *Add Device* and choose a name for your target device. In our example we choose TQM as target device name.

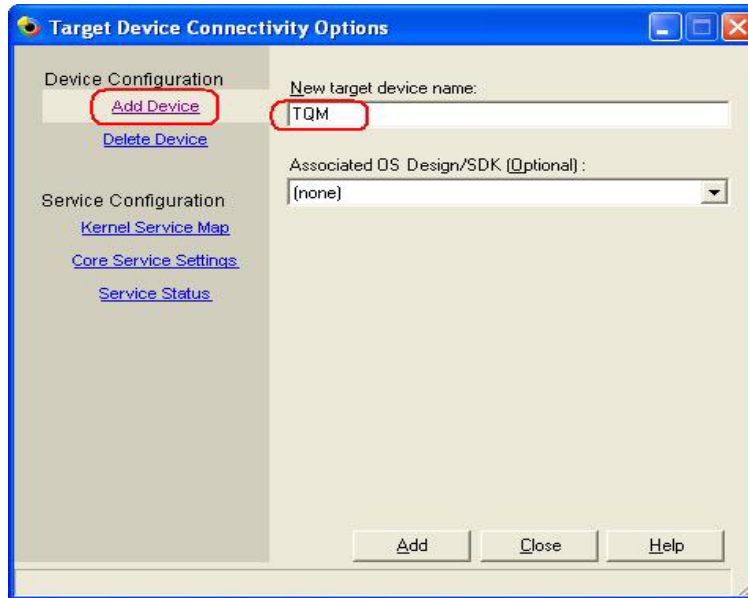


Figure 14: Add a new target device name

After you clicked *Add* the *Kernel Service Map* tab will be opened automatically:

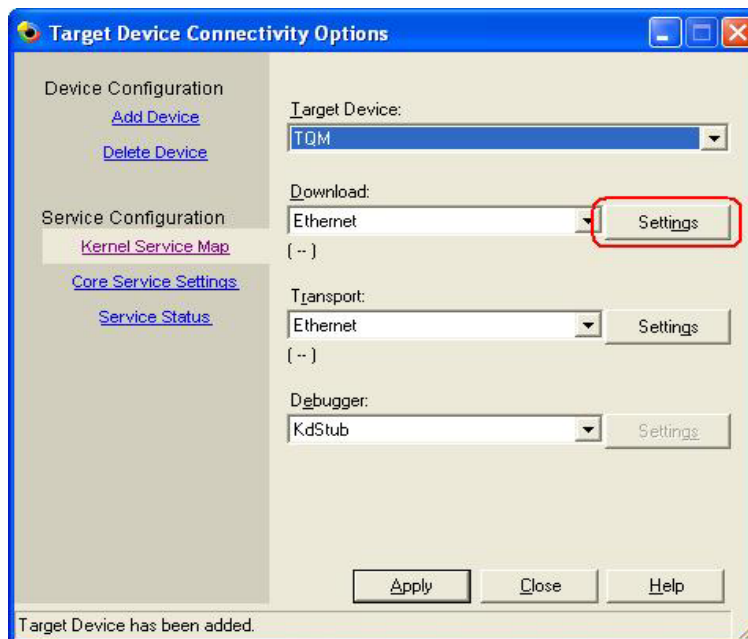


Figure 15: Target download settings

Make sure that *Ethernet* is selected for both, Download and Transport. Click the *Settings* button of the Download option. A new dialog will appear where you can choose the device to connect to. At the moment the *Active Devices* list will be empty. Now start your TQMa31 module. When the Startup menu appears, just press *Enter* to choose the download option.

EBOOT sends a broadcast message to the Platform Builder and your device will show up in the *Active Devices* list as MX31XXXXX (where XXXXX is a number depending on your network card's MAC address). If your device does not show up in the list of active devices, proceed to chapter [Debugging the download connection](#) and Configuration options of the EBOOT bootloader. It is possible, that the option *Autoboot* is not set to *Disabled*.

Select the device and click *OK*. Now your device is selected for both, Downloading and Debugging.

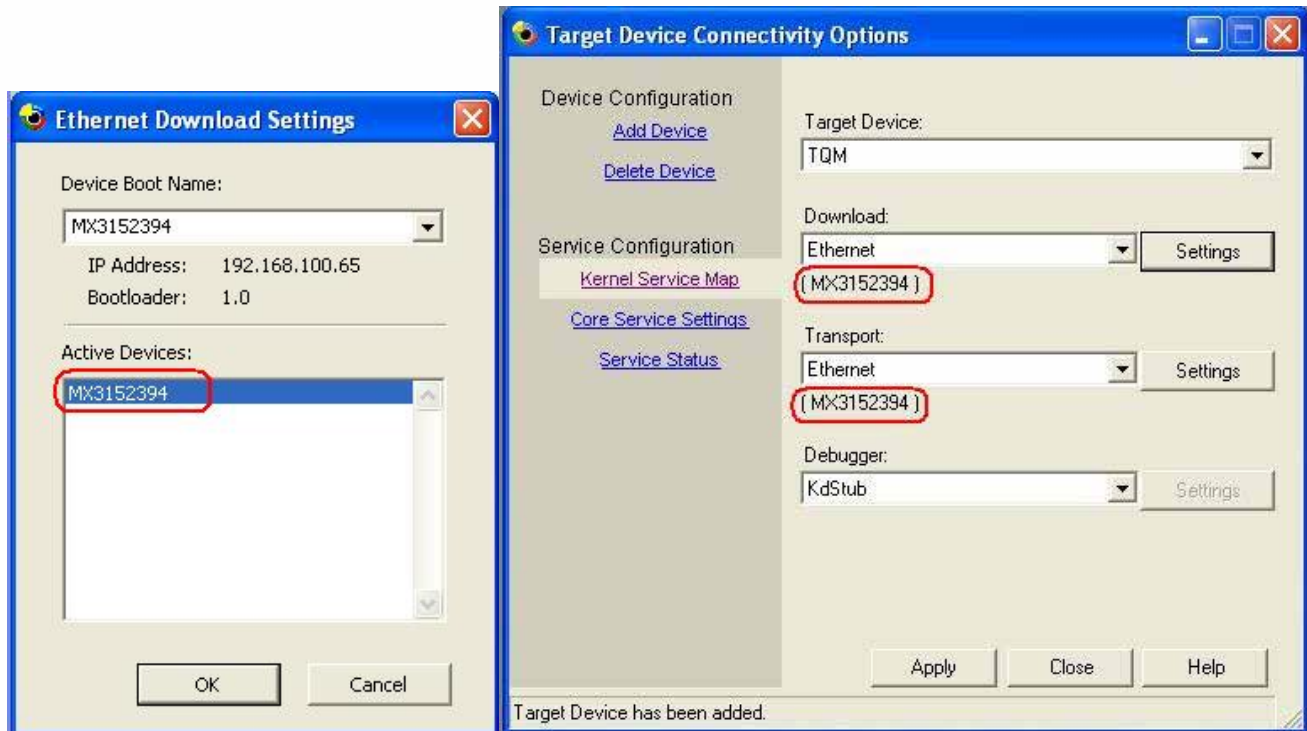


Figure 16: Selecting a active target device

Click *Apply* and then close the dialog.

Now you can select *Attach* from the *Target* menu in the Platform Builder IDE. A new dialog will appear and the download of the run-time image will start. If it is not, then reboot the TQMa31 module and wait for the download to begin.

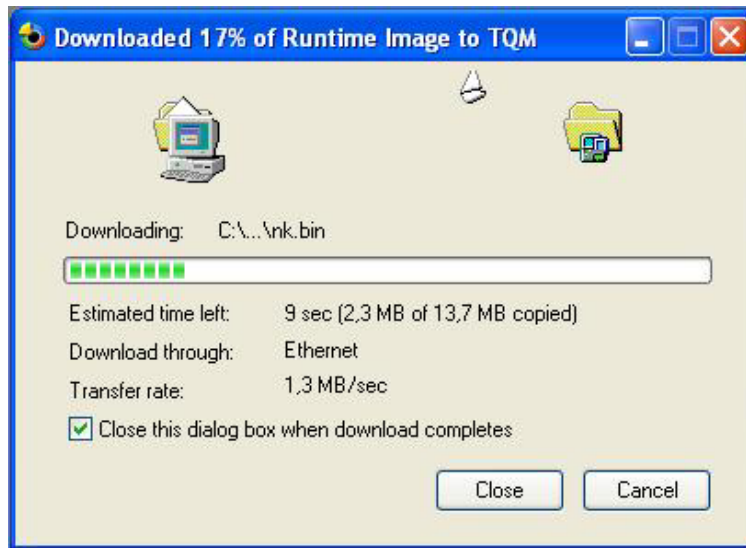


Figure 17: Downloading the Runtime Image

After the download is finished Windows CE should start up on your target device. For subsequent downloads just select again *Attach* from the *Target* menu (you don't need to configure the remote connection again unless you change targets).

5.5.2 Downloading a run-time image into SDRAM using EBOOT

To download a run-time image into the SDRAM of the target, follow these steps:

1. Open the desired workspace within Platform Builder.
2. Deselect **Platform** → **Settings** → **Build Options** → **Write Run-time Image to Flash Memory**.
3. Build the run-time image following the steps provided in [Building the OS Design](#).
4. Reset the TQMa31 module to launch EBOOT on the target.
5. If a target device connection has not been created within Platform Builder, follow the steps in [Configuring Ethernet Connection](#) to establish a connection.
6. From the Platform Builder *Target* menu, select *Attach Device* to begin the download.

5.5.3 Downloading a run-time image into NOR Flash using EBOOT

To download a run-time image into the NOR Flash of the target, follow these steps:

1. Open the desired workspace within Platform Builder.
2. Select **Platform** → **Settings** → **Build Options** → **Write Run-time Image to Flash Memory**.
3. Build the run-time image following the steps provided in Chapter [Building the OS Design](#).
4. Reset the TQMa31 module to launch EBOOT on the target.
5. If a target device connection has not been created within Platform Builder, follow the steps in [Configuring Ethernet Connection for Downloading and Debugging](#) to establish a connection.
6. From the Platform Builder *Target* menu, select *Attach Device* to begin the download.
7. After the download is complete, switch over to your terminal emulation application.
At this point EBOOT has downloaded the image temporarily into SDRAM and is ready to begin the flash programming procedure.

8. In the terminal emulation application, hit the 'y' key to begin programming the flash.
INFO: Downloading NK NOR image.
WARNING: Flash update requested.
Do you want to continue (y/n)?



Warning:
DO NOT SWITCH OFF THE POWER SUPPLY DURING FLASH UPDATE !

9. EBOOT will program the image into flash and provide status using serial debug messages.
Once the programming is complete, you will see the following messages:
INFO: Flashing sequence complete.
Reboot the device manually...
SpinForever...
10. To boot the programmed image automatically follow the Instructions given in Chapter [Running an OS Image from NOR Flash using EBOOT](#).

5.5.4 Downloading a run-time image running from SDRAM into NOR Flash using EBOOT (relocated Image)

If you want that the run-time image will be stored in NOR flash but that it will run out of RAM, follow these steps:

1. Open the desired workspace within Platform Builder.
2. Deselect **Platform** → **Settings** → **Build Options** → **Write Run-time Image to Flash Memory**.
3. Build the run-time image following the steps provided in Chapter [Building the OS Design](#).
2. Use the **File** menu of Platform Builder and choose **Open Workspace**.
3. Select NK.NB0 found in the workspace release directory:
WINCE500\PBWorkspaces\TQMa31_Workspace\RelDir\TQMa31_ARMV4I_Release\NK.nb0
4. Follow the steps in [Configuring Ethernet Connection for Downloading and Debugging](#) to establish an Ethernet connection between the target and Platform Builder.
5. From the Platform Builder *Target* menu, select *Attach Device* to begin the download.
6. Switch over to your terminal emulation application. You will be asked for the destination of the image. You will see the following message:
Specify image destination [0 = RAM, 1 = NOR, 2 = RAM image in NOR]:
7. In the terminal emulation application, hit the '2' key.
At this point EBOOT has downloaded the image temporarily into SDRAM and is ready to start the flash programming procedure. You will see the following message:
Storing RAM image into NOR
INFO: Downloading NK NOR image.
WARNING: Flash update requested.
Do you want to continue (y/n)?



Warning:
DO NOT SWITCH OFF THE POWER SUPPLY DURING FLASH UPDATE !

8. In the terminal emulation application, hit the 'y' key to begin programming the flash.
9. EBOOT will write the run-time image into flash and provide status using serial debug messages.
Once the programming is complete, you will see the following messages:
INFO: Flashing sequence complete.

Reboot the device manually...
SpinForever...

10. To boot the written run-time image automatically follow the Instructions given in Chapter [Running an OS Image from NOR Flash using EBOOT](#). The run-time image will be relocated from NOR to SDRAM before to start.

5.5.5 **Running a run-time image from NOR Flash using EBOOT**

To execute an OS image from NOR flash that has been previously programmed using the procedure described in [Downloading an OS Image into NOR Flash using EBOOT](#), follow these steps.

1. Reset the TQMa31 module to launch EBOOT on the target.
2. Quickly switch over to your terminal emulation application and wait for the debug message "Press [ENTER] to download now or [SPACE] to cancel." to appear.
3. Hit the space bar to bring up the EBOOT configuration menu.
4. Continue selecting the *Autoboot* option of the EBOOT menu until "NK from NOR" is selected.
5. Specify the desired boot delay using the *Boot Delay* menu option.
6. Save the configuration using the EBOOT menu.
7. Reset the TQMa31 module to launch EBOOT again. EBOOT will automatically jump to the run-time image in NOR flash after the specified boot delay.

5.5.6 **Debugging the download connection**

If you have problems with connecting your device through Platform Builder, you can observe through a serial connection if the initiation of the network card by the bootloader works well. In order to do so, connect the UART1 port of your TQMa31 module with a serial port of your host computer by means of a nullmodem cable. Use the program `C:\WINCE500\PUBLIC\COMMON\OAK\BIN\386\lcceterm.exe`, *Tera Term Pro* or *Hyperterminal* to open a connection at 115200/8/N/1.

```

Microsoft Windows CE Ethernet Bootloader Common Library Version 1.1 Built Aug 11 2008 10:39:25
Microsoft Windows CE Ethernet Bootloader 1.0 for TQMa31 (Aug 11 2008 10:40:03)
INFO: Bootloader launched from NOR
INFO: Reading boot configuration in NOR flash (addr = 0xA0020000, size = 0x60)
System ready!
Preparing for download...

Press [ENTER] to download now or [SPACE] to cancel.

Initiating image download in 3 seconds.
Initiating image download in 2 seconds.
Initiating image download in 1 seconds.
Initiating image download in 0 seconds.
Starting auto download ...

Initialize Ethernet controller.

Initialize Ethernet controller ETH_DEVICE_SMSC9118.
LAN9118: Chip Id 115a Revision 0
Auto Negotiation done
INFO: SMSC9118 Ethernet controller initialized.
INFO: MAC address: 0-12-33-55-CC-AA
INFO: Using device name: 'MX3152394'
InitDHCP(): Calling ProcessDHCP()
ProcessDHCP():DHCP_INIT
Got Response from DHCP server, IP address: 192.168.100.65

ProcessDHCP():DHCP IP Address Resolved as 192.168.100.65, netmask: 255.255.255.0
Lease time: 3600 seconds
Got Response from DHCP server, IP address: 192.168.100.65
No ARP response in 2 seconds, assuming ownership of 192.168.100.65
+EbootSendBootmeAndWaitForTftp
Sent BOOTME to 255.255.255.255

```

6 EBOOT Bootloader

6.1.1 Configuration options of the EBOOT bootloader

First you have to configure one COM port of your development computer with following parameters (115200/8/N/1) as shown:

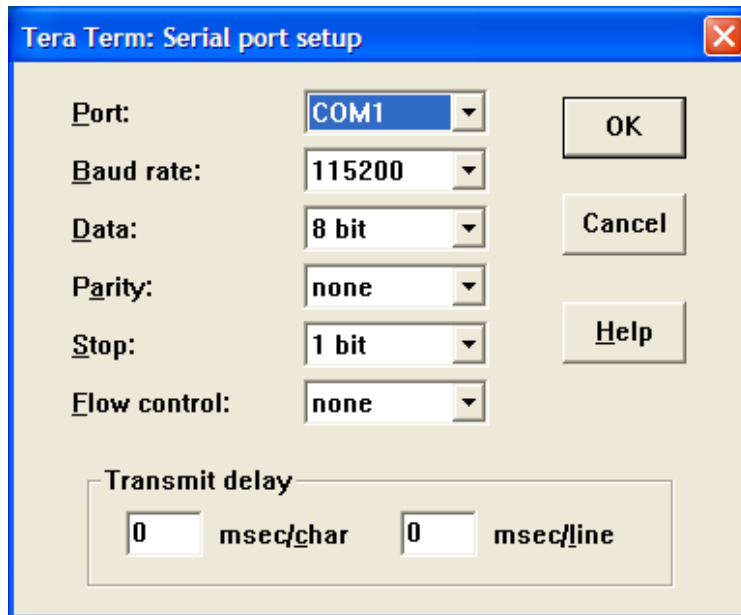


Figure 18: COM port configuration for EBOOT bootloader

After that, when the device is powered up and you have connected UART1 of port of your TQMa31 module with a serial port of your development computer by means of a nullmodem cable, you can enter the configuration menu by hitting the SPACE key in your terminal program when you see the following message:

Press [ENTER] to download now or [SPACE] to cancel.

Hitting SPACE will bring up the bootloader configuration menu:

Boot Loader Configuration:

- 0) IP address: 0.0.0.0
- 1) Subnet Mask: 0.0.0.0
- 2) Boot delay: 1 seconds
- 3) DHCP: Enabled
- 4) Reset to factory default configuration
- 5) Autoboot: NK from NOR
- 6) MAC address: 0-12-33-55-CC-AA
- 7) CPU speed: 532MHz AHB-Bus: 133MHz
- 8) L2 cache: on
- 9) KITL: passive
- S) Save configuration
- D) Download image now
- L) Launch existing flash resident image now

Enter your selection:

Here you can modify settings described in Table 2.

Option	Description
IP address	Select IP address for the bootloader (Not used if DHCP is enabled).
Subnet Mask	Select subnet mask for the bootloader (Not used if DHCP is enabled).
Boot delay	Delay in seconds that the bootloader waits for the SPACE key to enter in the configuration menu before the option selected in "Autoboot" is executed.
DHCP	Enable/disable DHCP for the bootloader
Reset to factory default configuration	Reset all setting to the factory defaults.
Autoboot	Select the boot option for normal startup (if not entered boot configuration). You can choose: NK from NOR Load run-time image from NOR flash NK from CFCard Load run-time image named NK.bin from CFCard. Disabled Bootloader will try to download image
MAC address	Select the MAC address for the onboard SMSC9215 network interface.
CPU speed	Select CPU speed. You can select either 532MHz or 399MHz. The AHB bus frequency will always be 133MHz.
L2 cache	Enable/Disable L2 cache.
KITL	Switch between active and passive KITL. Active KITL will try to connect a Platform Builder to establish a debug connection via Ethernet. Passive KITL will just print out the debug messages to UART1.
Save configuration	Saves the current configuration to NOR flash.
Download image now	Immediately start a download connection to load a run-time image from Platform Builder.
Launch existing flash resident image now	Launch the run-time image stored already in NOR flash.

Table 6-1: Bootloader options

6.1.2 Updating the EBOOT bootloader



WARNING:

Updating the EBOOT is dangerous. If the updated process does not finish successful, the EBOOT will not start anymore, and you will have to reprogram it again via JTAG debugger. Please contact TQ for more information.

Follow these steps to update the EBOOT image on the TQMa31 module:

1. Use the **File** menu of Platform Builder and choose **Open Workspace**.
2. Select the appropriate EBOOT.NB0 file you received from TQ.
Note:
The selection of the EBOOT.nb0 file (see folder \EBOOT on the CD) depends on the display (see [Direct Draw display driver](#)) and RAM configuration (see [Memory configuration](#)).
3. Follow the steps in [Configuring Ethernet Connection for Downloading and Debugging](#) to establish an Ethernet connection between the target and Platform Builder.
4. From the **Target** menu, select **Attach Device**.
5. After the download is complete, from the **Target** menu, select **Detach Device**.
6. Switch over your terminal emulation application. At this point, EBOOT has downloaded the image temporarily into SDRAM and is ready to begin the flash programming procedure.
7. In the terminal emulation application, hit the 'y' key to begin programming the flash.
8. EBOOT will program the image into flash and provide status using serial debug messages. Once the programming is complete, you will see the following messages:
INFO: Flashing sequence complete.
Reboot the device manually...
SpinForever...



Warning:

DO NOT SWITCH OFF THE POWER SUPPLY DURING FLASH UPDATE

9. Close the Platform Builder workspace for EBOOT.nb0. It is not necessary to save any workspace changes.
10. Reset the target hardware. If you see new EBOOT messages appear on the terminal, EBOOT has been properly programmed into NOR flash.

7 Device drivers

This chapter describes the individual settings for the included device drivers of the BSP.

Note:

If you want to create more than one workspace with different driver settings (e.g. one with DHCP and another one with static IP-Address) you will have to relocate the driver registry settings from the BSP (platform.reg) to your workspace (project.reg).

7.1 Display driver

7.1.1 Direct Draw display driver

At the moment the following displays are supported:

- ADMATEC 800x480 (NLC800T70D480CTMK)
- POWER TIP 320x240 (PH320240T-009-IC1-Q)

7.1.1.1 Adding the display driver to your OS design

Right-click on the „ADMATEC NLC800T70D480CTMK (800x480)“ or „POWER TIP (320x240)“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers -> Display**) and choose „Add to OS Design“.

Note:

Be sure, that the component *Direct Draw* (**Catalog -> Core OS -> Windows CE devices -> Graphics and Multimedia Technologies -> Graphics**) is included in your workspace.

7.1.1.2 Configuration of the direct draw display driver

Generally there is nothing to configure for the driver. If it is necessary to hide the mouse cursor, the user can set the following registry key in the project.reg.

```
[HKEY_LOCAL_MACHINE\System\GDI\Drivers]
;      0: Cursor Off, 1: Cursor On
      "CursorOn"=dword:1
```

Note:

The cursor off functionality only works, when the hive based registry is also added to your OS design (see Hive Based Registry). Turning the cursor on or off could be achieved by a small program which flushes the appropriate value for the above mentioned registry sub key „CursorOn“.

7.1.2 Mobile 3D display driver

At the moment the following displays are supported:

- POWER TIP 320x240 (PH320240T-009-IC1-Q)

Note:

The Mobile 3D (MBX) display driver is in a pre-release stage.

7.1.2.1 Adding the display driver to your OS design

Right-click on each of the components "MBX MX31 Base Driver", "MBX MX31 D3DM" and "MBX MX31 Ogles Core" in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers -> MBx**) and choose „Add to OS Design“.

Additionally you must add the component *Direct3D Mobile* (**Catalog -> Core OS -> Windows CE devices -> Graphics and Multimedia Technologies -> Graphics**) to your design.

7.1.2.2 Configuration of the mobile 3D display driver

There is nothing to configure for the driver.

7.2 VFP (Vector Floating Point) support

The VFP component adds support for the VFP coprocessor within the ARM11 processor to support floating-point arithmetic.

7.2.1 Adding VFP support to your OS design

Right-click on the „VFP“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers -> VFP**) and choose „Add to OS Design“.

7.2.2 VFP configuration

There is nothing to configure for this component.

7.3 Network

7.3.1 SMSC9215 Fast Ethernet driver

The SMSC9215 Fast Ethernet Driver adds support for the onboard network of your TQMa31 module.

7.3.1.1 Adding the SMSC9215 Network Driver to your OS design

Right-click on the „SMSC9215 NDIS“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQM -> Device Drivers -> Networking -> Local Area Networking (LAN) devices**) and choose „Add to OS Design“.

7.3.1.2 Network Configuration for the SMSC9215 Fast Ethernet Driver

The network driver configuration is done via registry settings. These settings are located in the platform.reg file.

Default setting for the network configuration is DHCP. If you have no DHCP server in your network or if you want to assign a static IP address for your network controller, add the following environment variable in the *Environment* tab of the *Platform Settings* dialog:

STATIC_IP=1

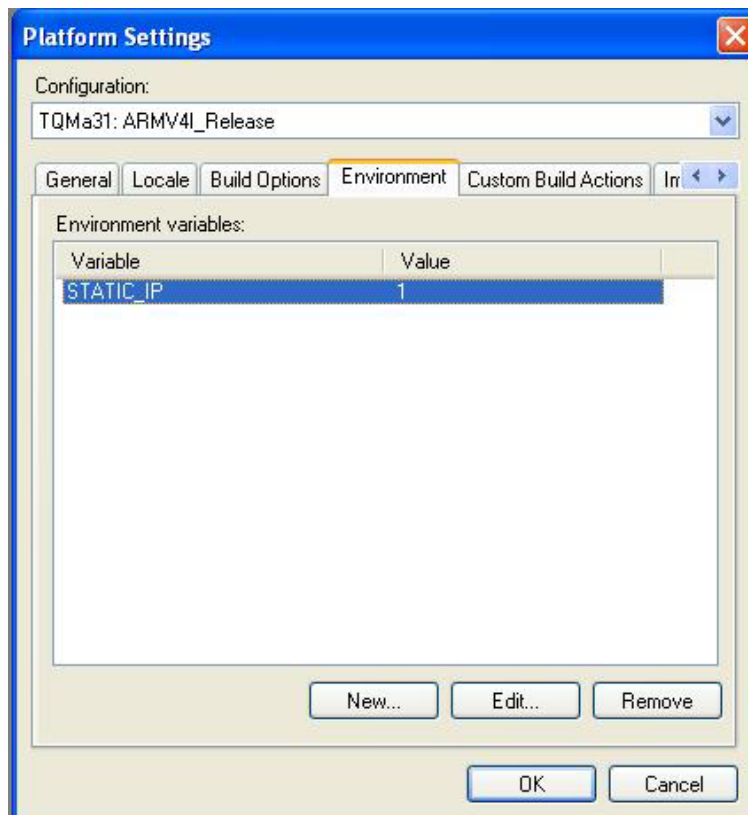


Figure 19: Create a environment variable for setting up a static ip

You can set the static IP address, DNS, Gateway... in the registry file of the driver (\$(_WINCEROOT)\TQMa31\FILES\platform.reg). The table 7-1 shows the possible values. Figure 27 shows an example configuration with static IP address.

Name	Values	Description
EnabledDHCP	0 or 1	Disable (value=0) or enable (value = 1) DHCP. If enabled, Windows CE tries to get its IP address from a DHCP server connected to your network.
DefaultGateway	"0" or "aaa.bbb.ccc.ddd"	The IP address of the default gateway.
IpAddress	"0" or "aaa.bbb.ccc.ddd"	If DHCP is used, the IP address should be set to "0.0.0.0". Otherwise you can assign a fixed IP address to your network adapter card here.
Subnetmask	"0" or "aaa.bbb.ccc.ddd"	The IP subnet mask.
DNS	"0" or "aaa.bbb.ccc.ddd"	The IP address of a DNS server.

Table 7-1: TCP/IP configuration Registry Keys

<pre>[HKEY_LOCAL_MACHINE\COMM\SMSC91181\Parms\TcpIp] "EnabledDHCP"=dword:0 "DefaultGateway"="192.168.100.1" "IpAddress"="192.168.100.25" "Subnetmask"="255.255.255.0" "DNS"="192.168.100.1"</pre>

Figure 20: Example for a static IP configuration

Note:

If KITL support is active the SMSC9215 driver will be replaced by the VMINI driver. Therefore TCP/IP has to be configured in the VMINI section of platform.reg.

7.3.2 IP network printer

To have support for IP printers, first add the component "PCL Printer Driver" (**Catalog -> Device Drivers -> Printer Devices**) to your OS design.

Furthermore TQ offers a SDK for .NET Compact Framework 2.0 from a third party, which offers you easily to implement printer functions within your application. Please see chapter PrinterCE.NETCF SDK for Visual Studio 2005 for more details.

Note:

The component "TCP/IP" (**Catalog -> Core OS -> Windows CE Devices -> Communication Services and Networking -> Networking Features**) should already be added to your OS design.

7.4 USB

7.4.1 USB OTG drivers

The USB OTG driver provides High Speed USB 2.0 host and device support for the USB “On The Go” (OTG) port of the TQMa31 module. The OTG driver will automatically select either Host (master) or Device (slave) functionality for high speed at any given time, depending on the USB cable/mini-plug configuration.

This is achieved by the set of three drivers: USB OTG host controller driver, USB client driver and/or USB transceiver controller (“Full Function”) driver, which performs the host/function client switching.

The USB host driver can be configured for class support for mass storage, HID, printer, and RNDIS peripherals. The device/client portion can be configured to provide one of mass storage, serial, or RNDIS function.

Note:

The USB OTG port can only be configured in one of the 3 possible device roles (Mass storage, Serial or RNDIS). The role is set with “DefaultClientDriver” value under the [HKEY_LOCAL_MACHINE\Drivers\USB\FunctionDrivers] key.

The “Full Function” OTG transceiver driver automatically selects between the host or client driver. The host or client can also be configured as the only mode for the OTG port, using the “Pure Host” or “Pure Client” catalog item. All the OTG catalog items are exclusive.

7.4.1.1 Adding the USB OTG drivers

Depending on the desired functionality right-click on the „Full OTG Function“, „Pure Client Function“ or “Pure Host Function” component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers -> USB Devices -> USB High Speed OTG Device -> *Desired Function***) and choose „Add to OS Design“.

7.4.1.2 USB OTG drivers configuration

There is nothing to configure for this driver.

7.4.2 USB high speed driver

7.4.2.1 Adding the USB high speed driver

Right-click on the „USB High Speed Host“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers -> USB Devices -> USB High Speed Host**) and choose „Add to OS Design“.

7.4.2.2 USB high speed driver configuration

There is nothing to configure for this driver.

7.4.3 USB Mouse and Keyboard support

To have support for USB keyboards and mice add the “USB HID Keyboard and Mouse” component to your OS design (**Catalog -> Core OS -> Windows CE devices -> Core OS Services -> USB Host Support -> USB Human Input Device (HID) Class Driver -> USB HID Keyboard and Mouse**).

7.4.4 USB printer support

To have support for USB printers add the components “USB Printer Class Driver” (**Catalog -> Core OS -> Windows CE devices -> Core OS Services -> USB Host Support**) and “PCL Printer Driver” (**Catalog -> Device Drivers -> Printer Devices**) to your OS design.

Furthermore TQ offers a SDK for .NET Compact Framework 2.0 from a third party, which offers you easily to implement printer functions within your application. Please see chapter PrinterCE.NETCF SDK for Visual Studio 2005 for more details.

7.5 SD Host Controller

The SD Host Controller Driver adds support for SD-Card functions of the i.MX31 CPU to your OS design.

7.5.1 Adding the SD Host Controller driver to your OS design

Right-click on the „SD Host Controller“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers -> SD Host Controller**) and choose „Add to OS Design“.

7.5.2 SD Host Controller driver configuration

There is nothing to configure for this driver.

7.6 Audio

The Audio Driver adds support for the audio functions of the i.MX31 CPU to your OS design.

7.6.1 Adding the Audio driver to your OS design

Right-click on the „Audio Driver“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers -> Audio**) and choose „Add to OS Design“.

7.6.2 Audio driver configuration

There is nothing to configure for this driver.

If you want to use the Windows CE system sounds or play wave (.wav) files, add the „Waveform Audio“ component to your OS design.

7.7 LM75 Temperature Sensor

The temperature sensor driver adds support for reading the temperature from 2 temperature sensors.

The Sensor 1 is located on the rear side of the TQMa31 CPU module, and the Sensor 2 is located on the base board.

Temperatures are given as 8bit signed values in units of 1°C.

Note:

Adding the temperature sensor driver also adds the I2C driver to your OS design.

7.7.1 Adding the temperature sensor driver to your OS design

Right-click on the „LM75 Temperature Sensor“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers**) and choose „Add to OS Design“.

7.7.2 Temperature sensor driver configuration

Normally there is nothing to configure for this driver.

If necessary, the I2C bus addresses of the sensors can be adjusted in the platform.reg file:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\TMP]
"Prefix"="TMP"
"Dll"="temp.dll"
"Index"=dword:1
"Order"=dword:5 ; load after I2C1
"LM75AddressCpuModule"=dword:48 ; 1. LM75
"LM75AddressBaseBoard"=dword:49 ; 2. LM75
```

7.7.3 How to use the temperature sensor driver

The LM75 temperature sensor driver supports the Windows CE standard stream driver interface.

A demo program **tempdemo.exe** is included in the BSP. To execute the demo please open a command window and type „tempdemo.exe“. The program will display the temperature values read from the LM75 temperature sensors.

The Sourcecode of this demo program can be found under
(\$_WINCEROOT)\PLATFORM\TQMa31\DRIVERS\TEMP\TMPDEMO.

7.8 Serial ports (UARTs)

Serial ports are used e.g. for ActiveSync or for connecting Touch-Screens to your system.

7.8.1 Adding serial drivers to your OS design

You can add up to three serial ports through the Platform Builder IDE.

To add support for COM1 right-click on the „UART1 serial port“ component in the Catalog (**Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> Serial**) and choose „Add to OS Design“. COM2 and 3 can be added to your OS design by selecting the „UART2 serial port“ and „UART3 serial port“ component.

7.8.2 Serial driver configuration

There is nothing to configure for this driver.

7.9 Touch panel support

The BSP includes a driver for the i.MX31 integrated 4 wire resistive touch controller.

7.9.1 Adding the touch panel driver

To add the touch panel driver right-click on the "Touch Driver" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> Touch)** and choose „Add to OS Design”.

7.9.2 Touch panel driver configuration

There is nothing to configure for this driver.

Note:

The default touch calibration data is set for displays with a resolution of 800x480 pixels in platform.reg file. Look for the [HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\TOUCH] section. For a individual calibration execute tohc.exe. and persist the [HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\TOUCH] key.

7.10 Dynamic Voltage and Frequency Control (DVFC)

The BSP includes a component called the Dynamic Voltage and Frequency Control (DVFC) driver that provides combined support for DVFS (Dynamic Voltage Frequency Scaling) and DPTC (Dynamic Process Temperature Compensation). The DVFC driver plays an important role in the reduction of active power consumption by dynamically adjusting the voltage and frequency settings of the system. The DVFC driver can respond to DVFC and DPTC hardware logic that is monitoring CPU loading and process/temperature performance of the silicon.

7.10.1 Adding the DVFC driver

To add the DVFC driver right-click on the "DVFC" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> DVFC)** and choose „Add to OS Design”.

Note:

The DVFC driver works only with a CPU frequency of 532MHz. The driver will not be loaded if CPU frequency is set to 399MHz.

7.10.2 DVFC driver configuration

There is nothing to configure for this driver.

7.11 I2C bus driver

The Inter-Integrated Circuit (I2C) module provides the functionality of a standard I2C slave and master. The I2C module is designed to be compatible with the standard Phillips I2C bus protocol.

7.11.1 Adding the I2C bus driver

To add the I2C driver right-click on the "I2C bus" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> I2C Bus)** and choose „Add to OS Design”.

7.11.2 I2C Bus driver configuration

There is nothing to configure for this driver.

7.12 Battery driver

The battery driver module is used to provide information about the battery level to the OS.

7.12.1 Adding the Battery driver

To add the Battery driver right-click on the "Battery Driver" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> Battery)** and choose „Add to OS Design”.

7.12.2 Battery driver configuration

There is nothing to configure for this driver.

7.13 Fake Battery Charger

This catalog item forces the battery driver to report a charger is connected and avoids the lowbattery condition. You should add this component if you don't have a battery powered device.

7.13.1 Adding the Fake Battery Charger

To add the Fake Battery Charger right-click on the "Fake Battery Charger" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> Charger)** and choose „Add to OS Design”.

7.13.2 Fake Battery Charger configuration

There is nothing to configure for this driver.

7.14 GPT driver

The General purpose timer is a multipurpose module used to measure intervals or generate periodic output.

7.14.1 Adding the GPT driver

To add the GPT driver right-click on the "GPT" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> Timers)** and choose „Add to OS Design”.

7.14.2 GPT driver configuration

There is nothing to configure for this driver.

7.15 IPU Backlight

The backlight driver interfaces with the Windows CE Power Manager to provide timed control over the display backlight. A timeout interval controls the length of time that the backlight stays on.

7.15.1 Adding the IPU Backlight driver

To add the IPU Backlight driver right-click on the appropriate driver component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> IPU Backlight)** and choose „Add to OS Design”.

7.15.2 IPU Backlight driver configuration

The following registry keys are required to properly load backlight driver.

```
[HKEY_CURRENT_USER\ControlPanel\Backlight]
    "BattBacklightLevel"=dword:7F ; Backlight level settings. 0xFF = Full On
    "ACBacklightLevel"=dword:AA ; Backlight level settings. 0xFF = Full On
    "BatteryTimeout"=dword:1E
    "ACTimeout"=dword:78
    "UseExt"=dword:1 ; Enable timeout when on external power
    "UseBattery"=dword:0 ; Enable timeout when on battery
    "AdvancedCPL"="AdvBacklight" ; Enable Advanced Backlight control panel dialog
```

The timeouts are given in seconds.

7.16 PC Card driver

The PC Card driver adds support for PCMCIA cards to your OS design.

7.16.1 Adding the PC Card driver

To add the driver right-click on the “MX31 PCMCIA (PC Card)” component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> PC Card -> PC Card Host)** and choose „Add to OS Design”.

7.16.2 PC Card driver configuration

There is nothing to configure for this driver. If you want to use CompactFlash memory cards in your PC Card socket add the ATADISK component to your OS design the **(Catalog -> Device Drivers -> Storage Devices -> Storage Devices -> Compact Flash / PC Card Storage (ATADISK))**.

If you insert a card into the socket, the device will show up as Hard Disk in explorer.

7.17 PWM driver

The Pulse Width Modulator (PWM) module is used to generate 16-bit resolution sound from sample audio images and can also be used to generate tones. It is used by the Buzzer component and therefore included automatically if you add the Buzzer component.

7.17.1 Adding the PWM driver

To add the driver right-click on the “PWM” component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> Pulse-width Modulator)** and choose „Add to OS Design”.

7.17.2 PWM driver configuration

There is nothing to configure for this driver.

7.18 RTC

The RTC driver adds the real-time clock support to your OS design.

7.18.1 Adding the RTC driver

To add the RTC driver right-click on the "RTC" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> RTC)** and choose „Add to OS Design“.

7.18.2 RTC driver configuration

There is nothing to configure for this driver.

7.19 Buzzer

The Buzzer driver drives the Buzzer pin of the TQMa31 module.

7.19.1 Adding the Buzzer driver

To add the Buzzer driver right-click on the appropriate driver component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> Buzzer)** and choose „Add to OS Design“.

7.19.2 Buzzer driver configuration

The Buzzer driver uses a standard Windows CE stream driver interface. The driver is configured via registry.

The Buzzer driver settings can be adjusted in the platform.reg file:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\BUZ]
"Prefix"="BUZ"
"Dll"="buzzer.dll"
"Index"=dword:1
"Order"=dword:2                ; load after PWM driver
"PulseWidth"=dword:32          ; 50%
"Frequency"=dword:3E8          ; 1kHz
"Duration"=dword:3E8           ; 1000ms
```

"PulseWidth" sets the pulse width of the signal in percent. Allowed values are from 1% to 100%.

"Frequency" sets the frequency of the signal in Hz. Allowed values are from 1Hz to 16kHz.

"Duration" sets the duration of the signal in milliseconds.

7.19.3 Buzzer driver usage

The following example code shows how to use the Buzzer driver. Each time you execute the WriteFile() command on the buzzer driver, the driver will play the sound described by the registry parameters.

```
// open Buzzer driver
HANDLE hBuzzer = CreateFile(L"BUZP1:",
```

```

        GENERIC_READ | GENERIC_WRITE, 0, NULL,
        OPEN_EXISTING, 0, NULL);
if(hBuzzer == INVALID_HANDLE_VALUE )
{
    printf("ERROR: Cannot open buzzer driver\n");
    return -1;
}
// play sound
WriteFile(hBuzzer, NULL, 0, NULL, NULL );

// wait 1 second
Sleep(1000);

// play sound again
WriteFile(hBuzzer, NULL, 0, NULL, NULL );

// close the Buzzer driver
CloseHandle(hBuzzer);

```

7.19.4 Buzzer test program

There is a test program for the buzzer included in the BSP. To add the program right-click on the „Buzzer Test Appliaction“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Demo Applications**) and choose „Add to OS Design“.

To execute the test, open a command window and enter “buztest.exe”. The test will play a sound with the parameters given in the registry.

7.20 International keyboard driver

As Windows CE natively only supports US, Japanese and Korean keyboards there are additional keyboard drivers for the most common European languages included in this BSP.

7.20.1 Languages included:

- German
- French
- Italian
- Spanish
- Dutch
- Swedish
- Portuguese

7.20.2 Adding the international keyboard drivers

Right-click on the „International Keyboard Drivers“ component in the Catalog (Catalog -> Third party -> BSPs -> TQM -> Device Drivers -> Input Devices -> International Keyboard Drivers) and choose „Add to OS Design“.

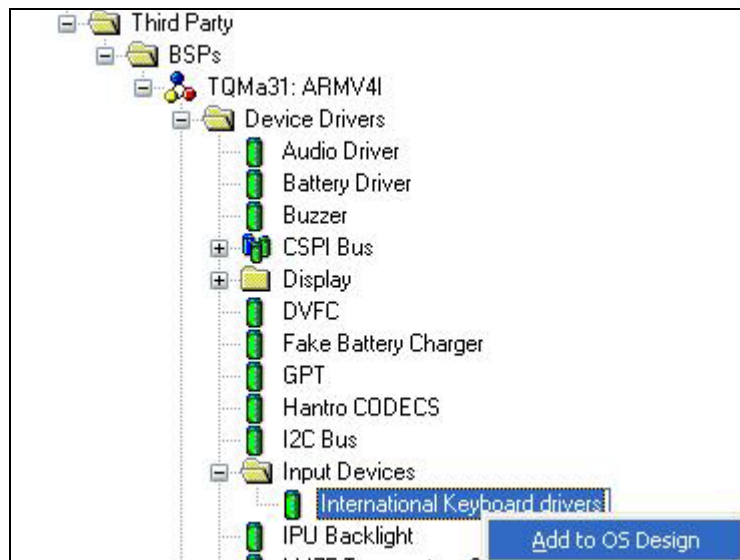


Figure 21: Adding the international keyboard drivers

7.20.3 Configuration of the international keyboard drivers

The keyboard driver configuration is done via registry settings. These settings are located in the kbd.reg file which is located in the \$(_WINCEROOT)\TQMa31\FILES directory of your BSP installation.

Each keyboard layout needs one folder located at [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Layouts] within the target's registry. This folder is named like the keyboard's language ID (e.g. "00000410" for Italian) and contains three keys:

- "Layout File"
Name of the keyboard driver DLL, e.g. "KbdIt.dll".
- "Layout Text"
Text to be displayed inside the language selection menu (e.g. "Italian") which is accessible via left click on the language tray icon in the task bar of the device, see figure below. You can set a user-defined text, regardless of the operating system's default language.

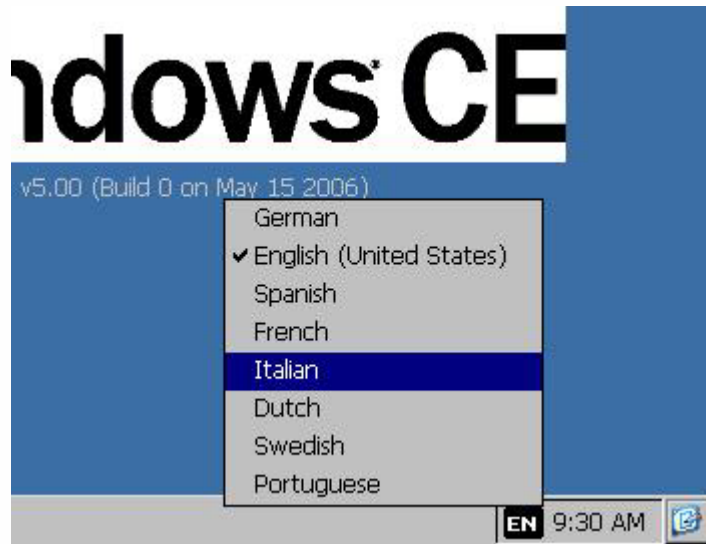


Figure 22: Setting the “Layout Text” key manually

The order of languages listed in the menu depends on the (hexadecimal) language ID, thus unchangeable.

- “KPPLayout”
Name of the keyboard driver DLL, the same as “Layout File”.
Additionally, each keyboard layout requires a unique preload value located at [HKEY_CURRENT_USER\Keyboard Layout\Preload]. The preload value is a consecutive number starting with 1 which is associated with every keyboard language ID used.

7.20.4 Example

The following example shows all registry settings required for the Italian keyboard layout:

```
[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Layouts\00000410]
  "Layout File"="KbdIt.dll"
  "Layout Text"="Italian"
  "KPPLayout"="KbdIt.dll"
[HKEY_CURRENT_USER\Keyboard Layout\Preload\3]
  @="00000410"
```

7.20.5 Setting the default keyboard layout

To define a default keyboard layout, set an *additional* preload key *without* trailed preload value and associate it with the language ID desired to be default. The following example shows the settings for Italian as default language:

```
[HKEY_CURRENT_USER\Keyboard Layout\Preload]
  @="00000410"
```

Furthermore the selected language must have the preload key with the trailed preload value of 1.

```
[HKEY_CURRENT_USER\Keyboard Layout\Preload\1]
@="00000410"
```

7.21 Matrix keypad driver

The TQMa31 module offers a 8x8 matrix keypad. Since the column and row lines are routed through the FPGA of the TQMa31 module, ghost keys could not be avoided by external hardware components (e.g. diodes). So you could only use the elements on the diagonal.

	1	2	3	4	5	6	7	8	
1	F1								Intended for a connection to a tactile switch emulates scan code for Functional Key 1
2		F2							Intended for a connection to a tactile switch emulates scan code for Functional Key 2
3			F3						Intended for a connection to a tactile switch emulates scan code for Functional Key 3
4				F6					Intended for a connection to an incremental encode (jog dial/wheel) emulates scan code for Functional Key 6
5					F7				Intended for a connection to an incremental encode (jog dial/wheel) emulates scan code for Functional Key 7
6						F5			Intended for a connection to an incremental encode (jog dial/wheel) emulates scan code for Functional Key 5
7									
8									

Figure 23: Matrix Keypad Layout

7.21.1 Adding the matrix keypad driver to your OS design

The matrix keypad driver will be automatically added to your OS design, when you add the international keyboard drivers to your OS design (see chapter [Adding the international keyboard drivers](#)).

7.21.2 Matrix keypad drive configuration

There is nothing to configure for this driver.

7.22 MCP2515 CAN driver

The MCP2515 CAN driver adds support to your OS design for the Microchip MCP2515 CAN controller located on the TQMa31 CPU module . This will also add the TQ_CanCe DLL to your OS design. The TQ_CanCe DLL acts as a user interface library for the MCP2515 CAN controller.

7.22.1 Adding the MCP2515 CAN driver to your OS design

Right-click on the „MCP1525 CAN driver“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers-r**) and choose „Add to OS Design“.

7.22.2 MCP2515 CAN driver configuration and usage

There is nothing to configure for this driver. All functions of the CAN interface are controlled via the TQ_CanCe interface dynamic link library.

For a detailed description of the user interface library (TQ_CanCe.dll) please see chapter CAN (TQ_CanCe.dll).

7.23 TQ CanCe Test Application

The TQ_CanCe DLL adds the interface library for the MCP2515 CAN controller to the OS design.

7.23.1 Adding the TQ CanCe Test Application to your OS design

Right-click on the „ TQ_CanCe Test Application“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Demo Applications**) and choose „Add to OS Design“.

7.23.2 TQ CanCe Test Application configuration

There is nothing to configure for this components. To start the application open a console window on your device and type “CanTest.exe”.

The Application is delivered with full source code to give an idea of how to use the TQ_CanCe interface library. The source is located in \$(_WINCEROOT)\PLATFORM\TQMa31\SRC\TEST\CanTest.

7.24 GPIO driver

The TQMa31GPIO driver adds support for 6 digital inputs and 8 outputs located on the TQMa31 CPU module to your OS design. This will also add the TQ_Gpio user interface DLL to your OS design.

7.24.1 Adding the GPIO driver to your OS design

Right-click on the „ TQMa31 GPIO Driver“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers**) and choose „Add to OS Design“.

7.24.2 GPIO driver configuration and usage

There is nothing to configure for this driver. All functions concerning the GPIOs are controlled via the TQ_Gpio interface dynamic link library (TQ_Gpio.dll).

For a detailed description of the user interface library, please see chapter General Purpose Input/Output (TQ_gpio.dll).

7.25 TQ Gpio Test Application

The Test application gives an example of how to use the GPIO API of the TQ_Gpio.dll.

Gpiotest is a command line application that shows how to use the TQ_Gpio.dll and call the functions of the GPIO driver. It allows to test the GPIO inputs and outputs.

When the application is started without parameters, it shows the help of the command line.

```
\> gpiotest

TQ GPIO .Net Test Application - TQ Systems (C) Nov 2008

usage: gpiotest [gpo:output:frequency] [gpi:input:mode] [t:seconds]

where: [gpo:output:frequency] set to high the bitwise selected gpo
      output is a 2 digits hexadecimal value where MSB = GPO_8; LSB = GPO_1
      frequency is a 4 digits value to set the frequency will be generated
      on the selected outputs

      [gpi:input[:mode]] counts the generated interrupts in the bitwise
      selected gpi
      input is a 2 digits hexadecimal value where MSB = GPI_5; LSB = GPI_1
      mode is the form in which the selected interrupts will be activate
          0 - Level Low Active
          1 - Level High Active
          2 - Rising Edge (default)
          3 - Falling Edge

      [t:seconds] seconds that the application waits for the inputs
          30 - waits 30 seconds (default)
```

\>

Example:

```
\> gpiotest gpi:0f:3
```

This sample starts one task for each selected interrupt input (GPI_1 – GPI_4 => 00001111b = 0x0f). These tasks show the state of the appropriate input data after the interrupt event will be started according of the selected interrupt mode.

```
\> gpiotest gpo:f0
```

This sample sets the selected outputs GPO_5 – GPO_8 to a logic high and GPO_1 – GPO_4 to low.

7.25.1 Adding the TQ Gpio Test Application to your OS design

Right-click on the „TQ_Gpio Test Application“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Demo Applications -> TQ_Gpio Test Application**) and choose „Add to OS Design“.

7.25.2 TQ Gpio Test Application configuration

There is nothing to configure for this components. To start the application open a console window on your device and type “GpioTest.exe”.

The Application is delivered with full source code to give an idea of how to use the TQ_Gpio interface library. The source is located in \$(_WINCEROOT)\PLATFORM\TQMa31\SRC\TEST\GpioTest.

7.26 Watchdog driver

The TQMa31Watchdog driver adds support for the i.MX31 watchdog to your OS design.

7.26.1 Adding the Watchdog driver to your OS design

Right-click on the „TQMa31 Watchdog Driver“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Device Drivers -> TQMa31 Watchdog Driver**) and choose „Add to OS Design“.

7.26.2 Watchdog driver configuration and usage

The Watchdog driver is configured by registry settings located in the platform.reg file:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\TQMA31_WDG]
"Dll" = "TQMA31_WDG.Dll"
"Prefix" = "WDG"
"Order" = dword:0
"FriendlyName" = "TQMa31 Watchdog Driver"
"StartupDelay" = dword:BB8 ; in ms (3000ms)
"Interval" = dword:BB8 ; in ms (will be rounded to 500ms steps)
"Type" = dword:1 ; 0: A driver thread triggers the watchdog (Automatic startup)
; 1: Application triggers the watchdog
"StartType" = dword:1 ; 0: Automatic startup
; 1: First application trigger will start the watchdog
```

Name	Values	Description
Type	0 or 1	0: The driver itself will start a thread which retriggers the watchdog every Interval/4 milliseconds. 1: A user application has to retrigger the watchdog (default).
StartType	0 or 1	0: Automatic startup. The watch dog is started automatically when the driver is loaded (This is also set when "Type" is set to 0) 1: The first trigger call (WriteFile()) will start the watchdog.
StartupDelay	Delay in ms.	Sets the initial delay before the Watchdog is initialized.
Interval	Interval in ms	Sets the watchdog timeout interval. If the watchdog is not retriggered before the timeout expires, the watchdog will reset the system. The value is rounded to full 500ms steps. Minimum Value 500ms, maximum is 128000ms.

Table 7-2: Watchdog driver registry settings.

By default the watchdog is configured for a 3 second timeout and usage by a application. All functions of the Watchdog interface are controlled via the TQ_Wdg interface dynamic link library.

The usage of the Watchdog driver is very straightforward and controlled by only 3 API calls:

```
/******
```

```

**
**      Function      : TQwdg_Open (mandatory)
**
**      Description   : Request access to the watchdog
**
**      Returnvalue    : >=0 - Driver handle
**                      TQc_BUSY if another program has init access
**                      TQc_LOADDRIVER if hardware driver not installed
**
**      *****/
TQCAPI TQwdg_Open(void);

/*****
**
**      Function      : TQwdg_Close
**
**      Description   : Close the driver
**
**      Parameters    : board_hdl (in) - driver handle.
**
**      Returnvalue    : TQc_OK if OK
**                      TQc_PARAMETER if invalid params passed
**
**      *****/
TQCAPI TQwdg_Close(UINT32 board_hdl);

/*****
**
**      Function      : TQwdg_Trigger
**
**      Description   : Retrigger the watchdog timer.
**                      The first call to TQwdg_Trigger will start the watchdog if
**                      "StartupType"=1 is set.
**
**      Parameters    : board_hdl (in) - driver handle.
**
**      Returnvalue    : TQc_OK if OK
**                      TQc_PARAMETER if invalid params passed
**
**      *****/
TQCAPI TQwdg_Trigger(UINT32 board_hdl);

```

The TQ_Wdg Test application shows the usage of the API.

7.27 TQ Wdg Test Application

The TQ_Wdg Test application gives an example of how to use the Watchdog driver.

7.27.1 Adding the TQ Wdg Test Application to your OS design

Right-click on the „TQ_Wdg Test Application“ component in the Catalog (**Catalog -> Third party -> BSPs -> TQMa31 -> Demo Applications -> TQ_Wdg Test Application**) and choose „Add to OS Design“.

7.27.2 TQ Wdg Test Application configuration

There is nothing to configure for this components. To start the application open a console window on your device and type "WdgTest.exe". The application will start the watchdog and retrigger it every second. When you end the application by hitting a key, the watchdog will expire after the Interval (set in the watchdog driver registry settings) is expired. The system will then reboot.

The Application is delivered with full source code to give an idea of how to use the watchdog driver. The source is located in \$(_WINCEROOT)\PLATFORM\TQMa31\SRC\TEST\WdgTest.

7.28 RNGA

The RNGA component adds the random number generator of the i.MX31 CPU to your OS design.

7.28.1 Adding the RNGA driver

To add the RNGA component right-click on the "RNGA" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> RNGA)** and choose „Add to OS Design”.

7.28.2 RNGA configuration

There is nothing to configure for this component.

7.29 NLED

The NLED driver adds support for the notification LED function to your OS design.

7.29.1 Adding the NLED driver

To add the RNGA component right-click on the "NLED" component in the **(Catalog -> Third Party -> BSPs -> TQMa31 -> Device Drivers -> NLED)** and choose „Add to OS Design”.

7.29.2 NLED configuration

The NLED driver uses a configurable memory address to control the notification LED via memory mapped IO. This memory address has to be set in the nleddrv.reg file in the \$(_WINCEROOT)\PLATFORM\TQMa31\SRC\DRIVERS\NLEDDRV\ directory.

Note:

Please consult with TQ Components before using this driver.

8 Developing Applications

The following steps guide you through the creation of an SDK for your OS design and shows you how to use eVC++ and Visual Studio 2005 to develop, debug and download applications.

8.1 SDK for eVC++

An SDK (Software Development Kit) for eVC++ enables you to develop applications for your OS design with the embedded VisualC++ 4.0 IDE. The SDK includes all the library and associated header files specific to your OS design.

8.1.1 Creating an SDK from your OS design

Note:

Before building the SDK please do a Sysgen of a Release Run-Time Image of your OS Design.

Start the Platform Builder IDE, open your workspace and select “New SDK...” from the “Platform” menu.

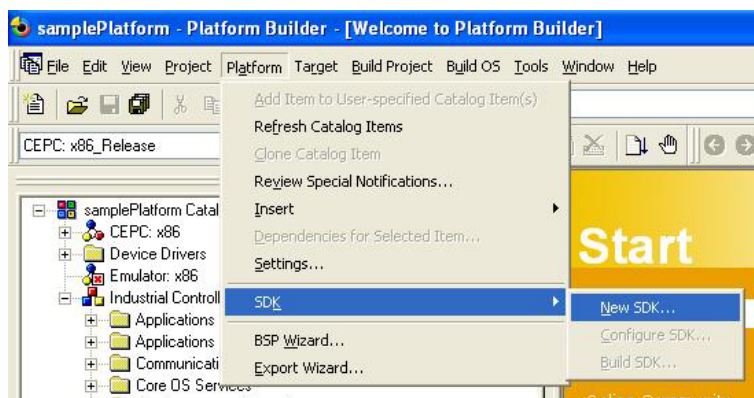


Figure 24: Create a new SDK for eVC++

Choose a unique name for the SDK, enter the name of your company and hit *Next >*.

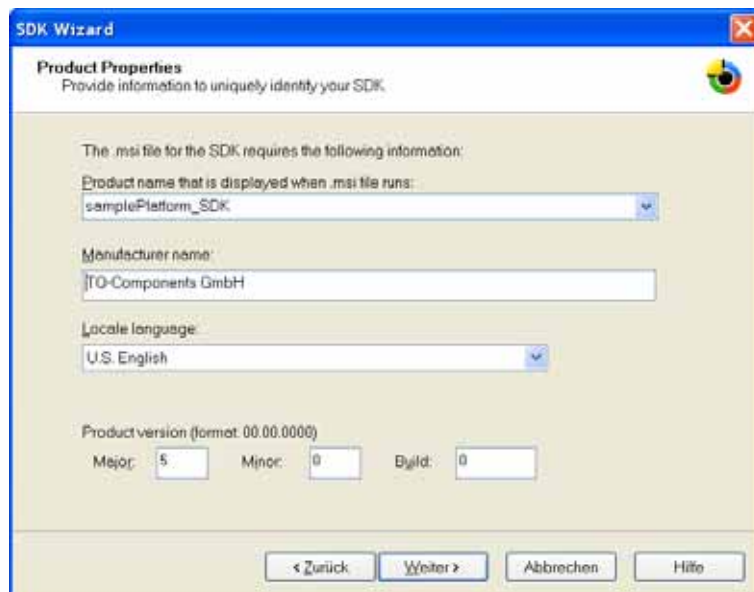


Figure 25: Setting up the SDK properties

After you click *Next >* you can select the development languages you want to support. Hit *Next >* and then *Finish*.



Figure 26: Selection of the development languages

To finally build the SDK select *Build SDK...* from the *Platform* menu.

Your SDK will be created as an MSI file located at: `$( _PBWORKSPACEROOT ) \SDK\`. It is self-installable by double clicking it and it is removable using “Add/Remove Programs” from the Windows XP Control Panel.

8.2 SDK for .NET Compact Framework

With the SDK for .NET compact framework you are able to develop applications in C# or Visual Basic with the Visual Studio 2005 development environment from Microsoft. The SDK includes all the libraries and associated header files specific of your OS design.

Note:

Before you create the SDK, you must add the support for the appropriate components for .NET compact framework. Platform builder gives you the choice between components for .NET Compact Framework 1.0 or 2.0.

- Therefore right-click on the component „ .NET Compact Framework 1.0/2.0 “ in the Catalog (**Catalog -> Core OS -> Windows CE devices -> Applications and Services Development -> .NET Compact Framework 1.0/2.0 -> .NET Compact Framework 1.0/2.0**), “OS Dependencies for .NET Compact Framework 1.0/2.0” in the Catalog (**Catalog -> Core OS ->**

Windows CE devices -> Applications and Services Development -> .NET Compact Framework 1.0/2.0 -> OS Dependencies for .NET Compact Framework 1.0/2.0) and choose „Add to OS Design“.

- Additionally add the “Standard SDK for Windows CE” component in the catalog (**Catalog -> Core OS -> Windows CE devices -> A Applications and Services Development**) to the OS design.

Afterwards you have to build the OS design (see Building the OS Design). After that, you can proceed with following chapter creating a SDK.

8.2.1 Creating a SDK from your OS design

Start the Platform Builder IDE, open your workspace and select “New SDK...” from the “Platform” menu.

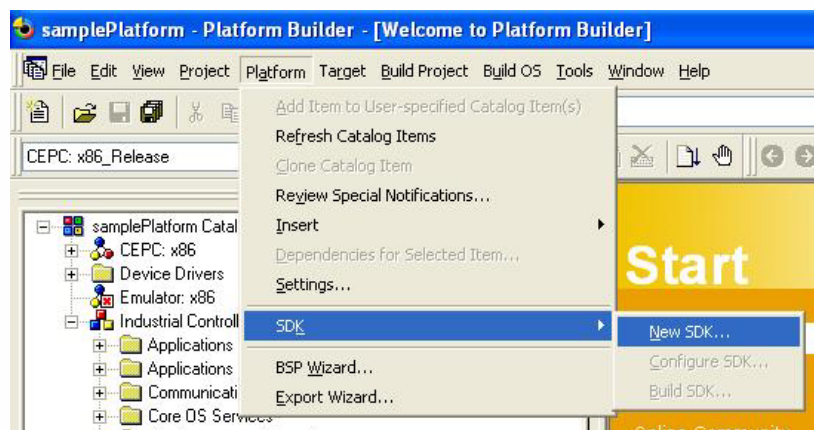


Figure 27: Create a new .NET CF SDK

Choose a unique name for the SDK, enter the name of your company and hit Next >.



Figure 28: Setting up the SDK properties

After you click *Next >* you can select the development languages you want to support. Here you must select the check box “.NET Compact Framework” and Hit *Next >* and then *Finish*.

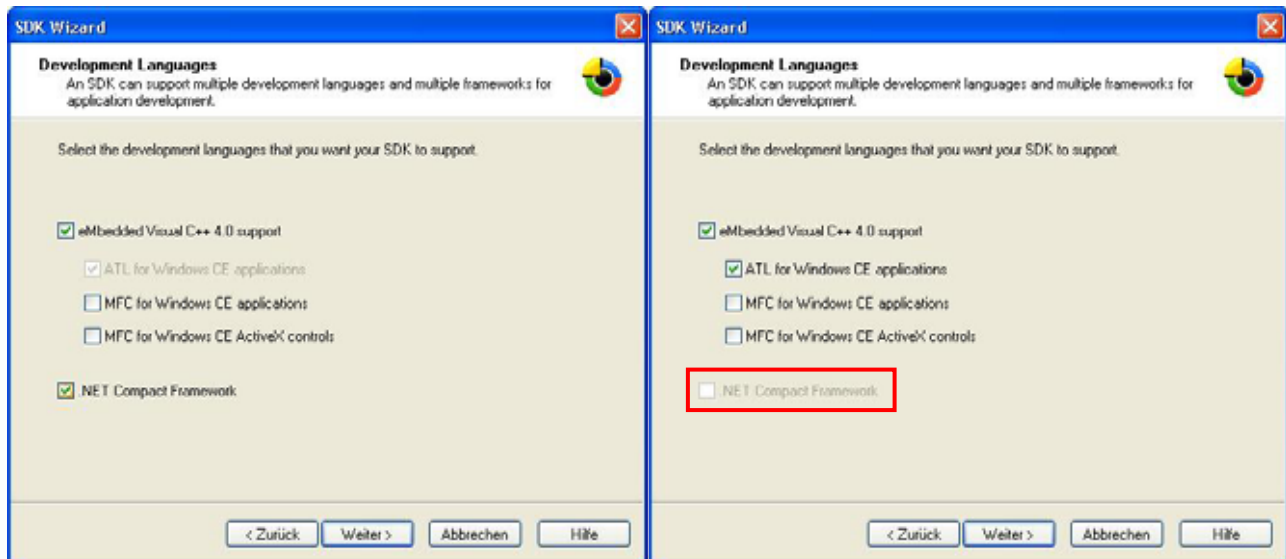


Figure 29: Selection of the development languages and behavior of the appropriate check boxes

Note:

If you did not add the “Standard SDK for Windows CE” component to the OS design, you are not able to select the check box, in this case the check box is grayed out.

Next you have to configure the SDK by selecting *Configure SDK...* from the *Platform* menu. First select the SDK target directory path, the OS design name and die MSI file name.

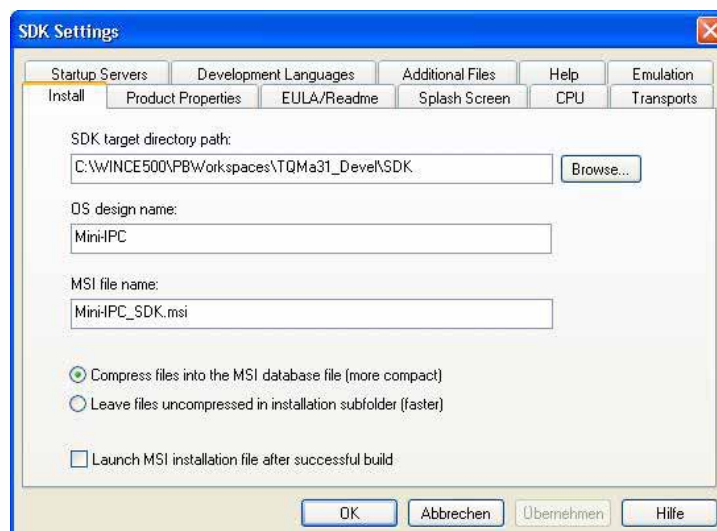


Figure 30: Modifying SDK properties

Then select the available startup servers and transports for debugging. Set up the appropriate default startup server and default transport and accept the new SDK settings.

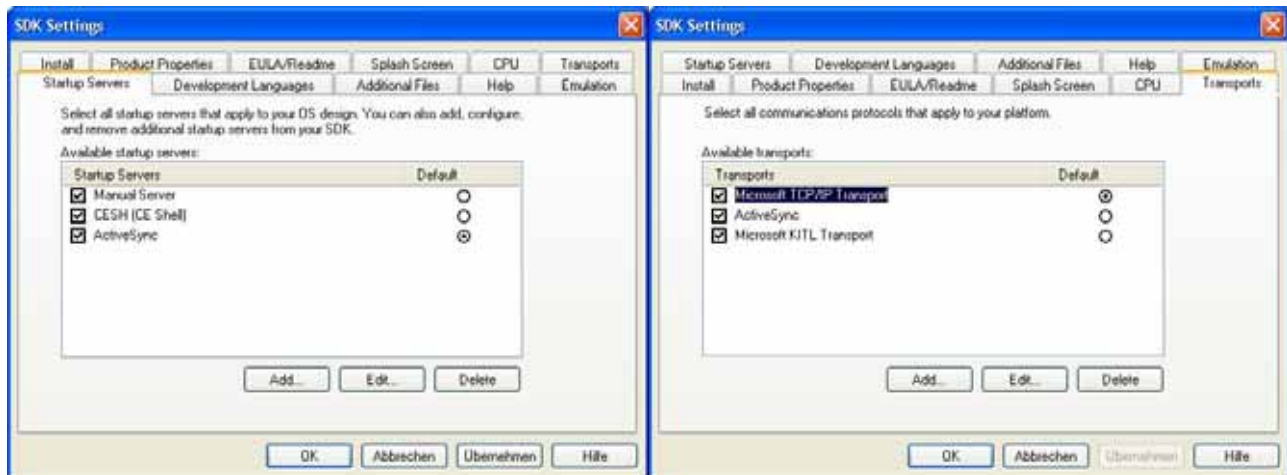


Figure 31: Setting up the startup servers and transports

Note:

After you have build a SDK with .NET Compact Framework successfully, you can remove the "Standard SDK for Windows CE" component from your OS design. Afterwards you will find the checkbox for .NET Compact Framework grayed out, but selected (**SDK settings -> tab "Development Languages"**).

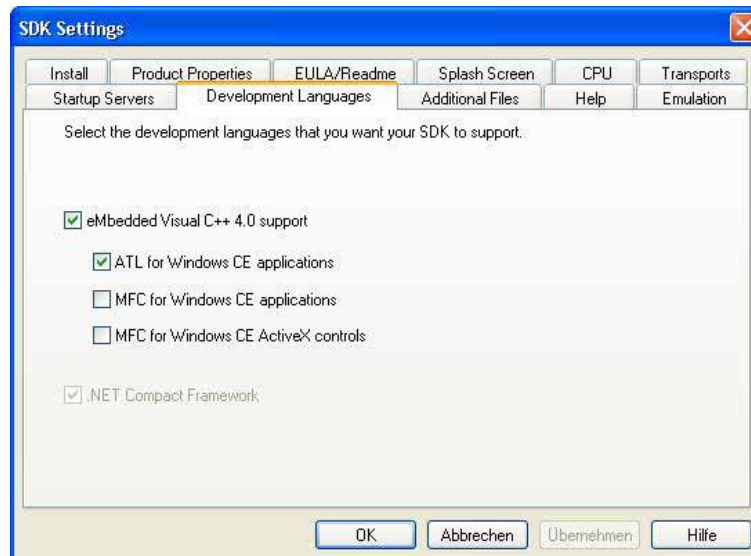


Figure 32: Selecting .NET CF support without Standard SDK support

To finally build the SDK select *Build SDK...* from the *Platform* menu. After the SDK is build successful, you will find the installable SDK-file (file extension msi) in your workspace directory.

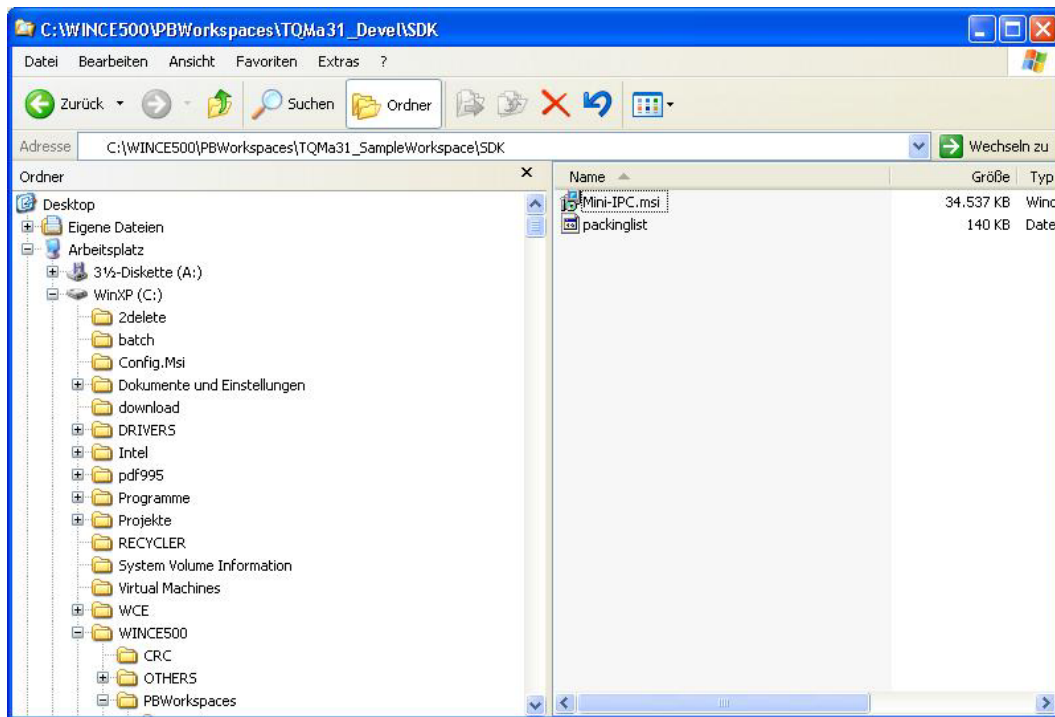


Figure 33: SDK target directory

8.3 Developing with embedded Visual C++

To develop and debug programs written in C/C++ to run with Windows CE, you can use Microsoft embedded Visual C++® or Microsoft Visual Studio 2005. In this chapter will be described the use of MS embedded Visual C++.

If you are familiar with Microsoft Visual Studio and Visual C++®, you may find that the graphical user interface for the embedded development environment has a nearly identical *look and feel*.

8.3.1 Installation

If your installation CD includes Service Pack 4 (SP4) assure that it will be installed as well. You will be prompted to select the installation including the SP4 it at the beginning of the installation.

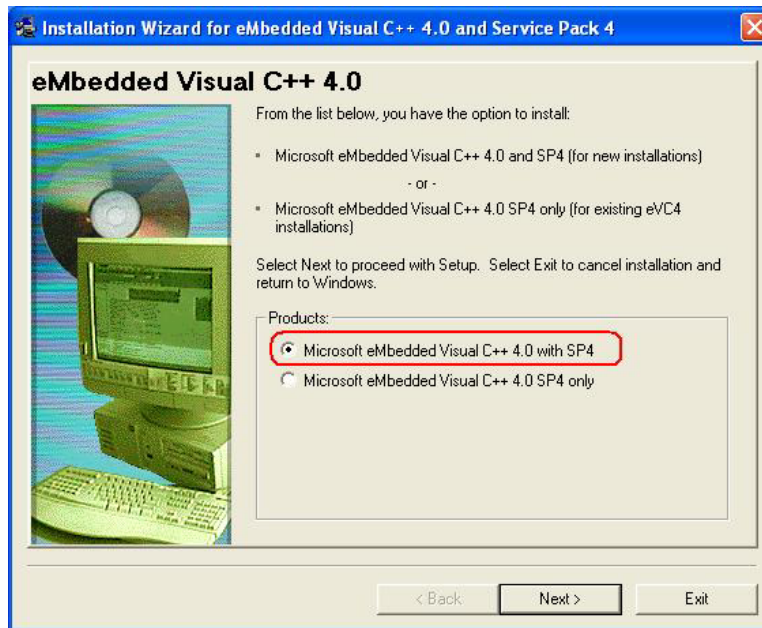


Figure 34: Installing eVC++ with SP4

If you already have eVC installed without Service Pack 4 you must download it from the Windows Embedded Developer Center in the Microsoft msdn web page.

During the installation of MS embedded Visual C++ (eVC) one reboot is needed. Leave the CD in the drive while Windows is rebooting because the installation continues after reboot.

8.3.2 Establish a connection between eVC and the CE device

Open embedded Visual C++ and select "New..." from the File menu.

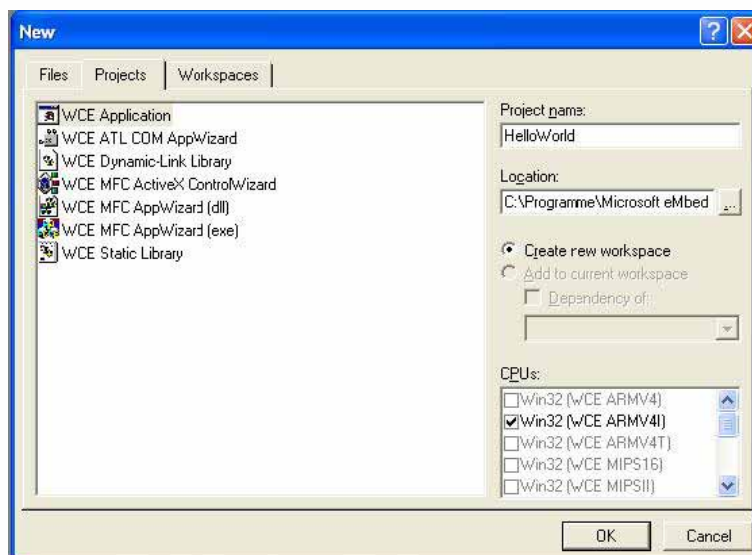


Figure 35: Create a new project for the ARMV4I CPU

In this example you may create a typical “Hello World” application for ARM4VI.

If your OS design SDK was successfully installed, you can select the OS design and the appropriate device as shown in the following example (samplePlatform):

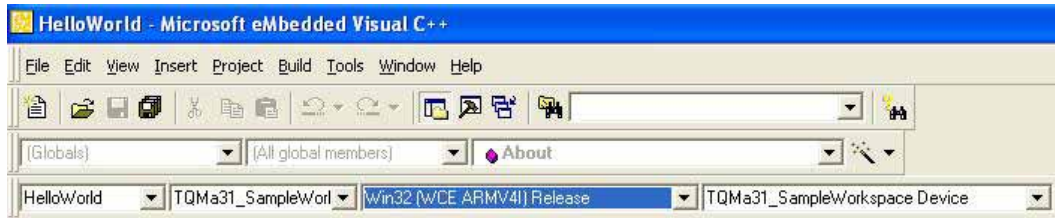


Figure 36: Selecting the appropriate device (depends on installed SDKs)

Now you have to configure your OS design. Go to the *Tools* menu and select *Configure Platform Manager...* . A dialog opens where you should select the SDK and the appropriate device.

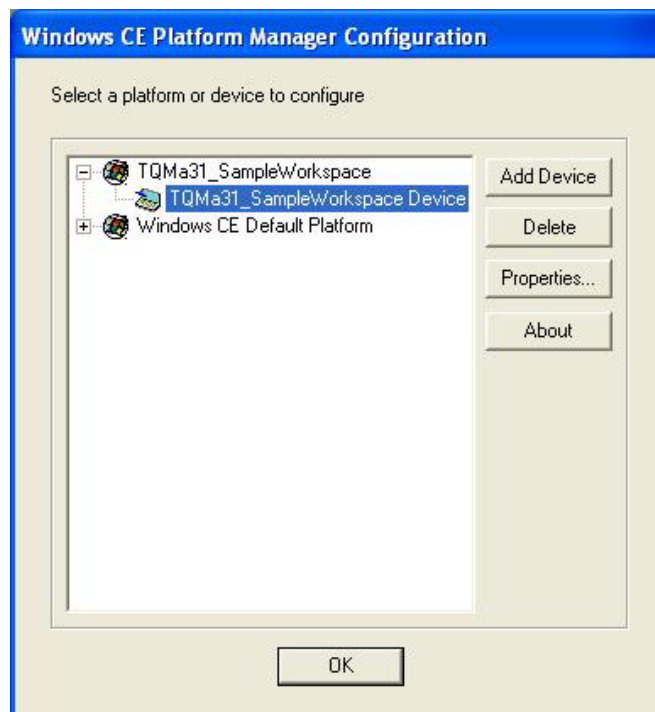


Figure 37: Configure Platform Manager

Go to *Properties...* to make connection settings.

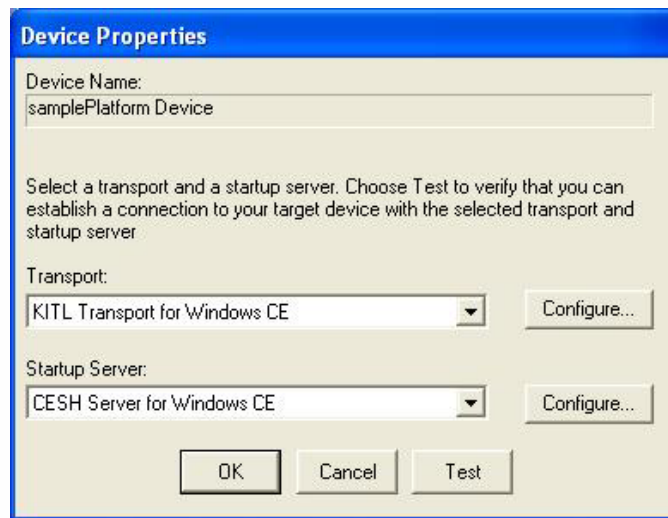


Figure 38: Configure Transport and Startup Server

If you have set up an Active Sync connection you can set Active Sync for Transport. For KITL Transport a connection between Platform Builder and the target must be established.

After successfully testing the connection, you should be able to compile your project and automatically download it to the device.

8.4 **Developing with Visual Studio 2005**

To develop and debug programs written in C++, C# and Visual Basic to run with Windows CE, you can use Microsoft Visual Studio 2005®.

8.4.1 **Installation**

Additionally to the installation DVD of Microsoft Visual Studio 2005 the following software components must be on-hand or installed:

- Microsoft .NET Framework 1.1
- Microsoft .NET Framework 2.0
- Service Pack 1 for Microsoft Visual Studio 2005
- Microsoft .NET Compact Framework 2.0
- Service Pack 1 for Microsoft .NET Compact Framework 2.0
- Service Pack 2 for Microsoft .NET Compact Framework 2.0
- ARMV4I specific SDK for .NET Compact Framework

Steps for installation:

Note:

If one or more of the following software components are already installed, please skip the appropriate installation:

1. Microsoft .NET Framework 1.1
2. Microsoft .NET Framework 2.0
3. Microsoft Visual Studio 2005
4. Service Pack 1 for Microsoft Visual Studio 2005
5. Microsoft .NET Compact Framework 2.0
6. Service Pack 1 for Microsoft .NET Compact Framework 2.0
7. Service Pack 2 for Microsoft .NET Compact Framework 2.0
8. ARMV4I specific SDK for .NET Compact Framework

8.4.2 Establish a connection between VS 2005 and the CE device

8.4.2.1 Startup over Active Sync / Transport over Ethernet

Note:

Before you start with the following steps, be sure that the Run-Time-Image on the target device has build in the active sync support (see chapter Establish a ActiveSync connection over USB).

Click on “Device Options” button on toolbar.

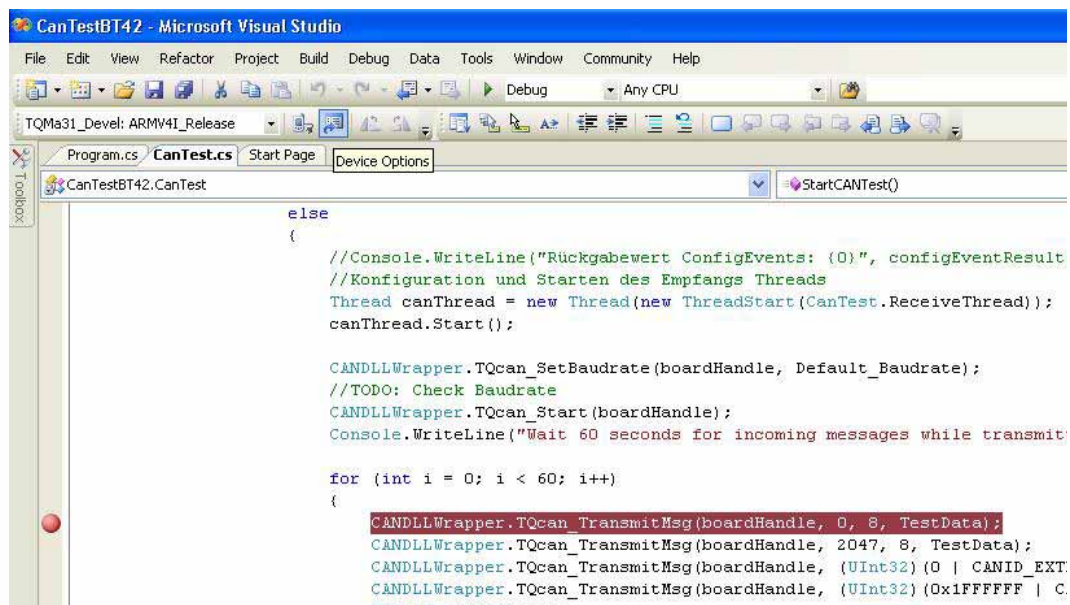


Figure 39: Device Options

Select the SDK OS design from “Show devices for platform”.

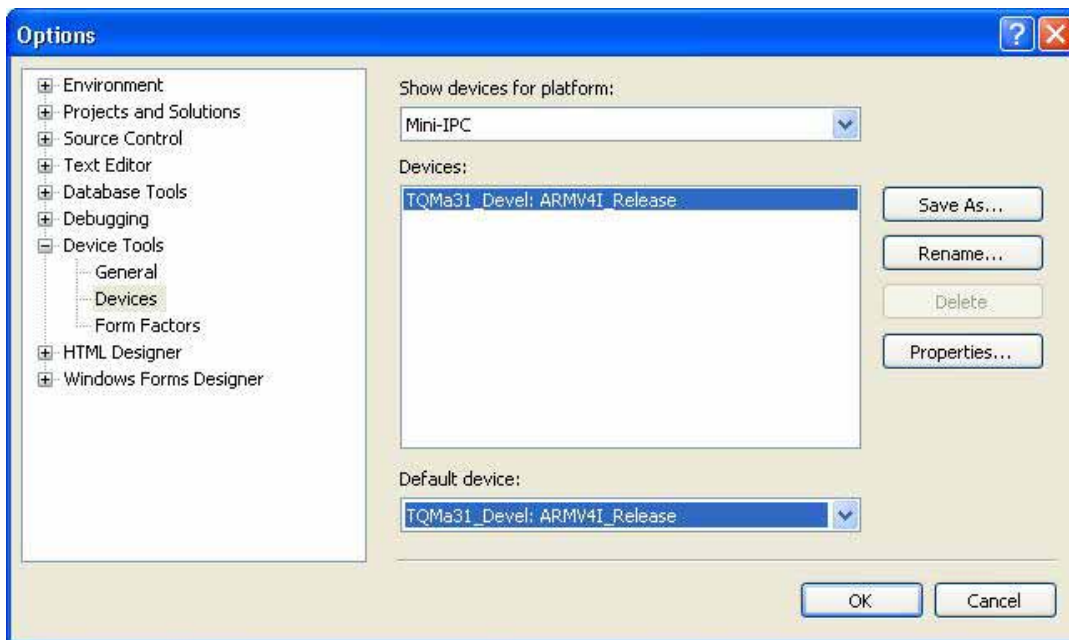


Figure 40: Selection platform device

Configure the IP address of the target (**Properties -> Configure**).

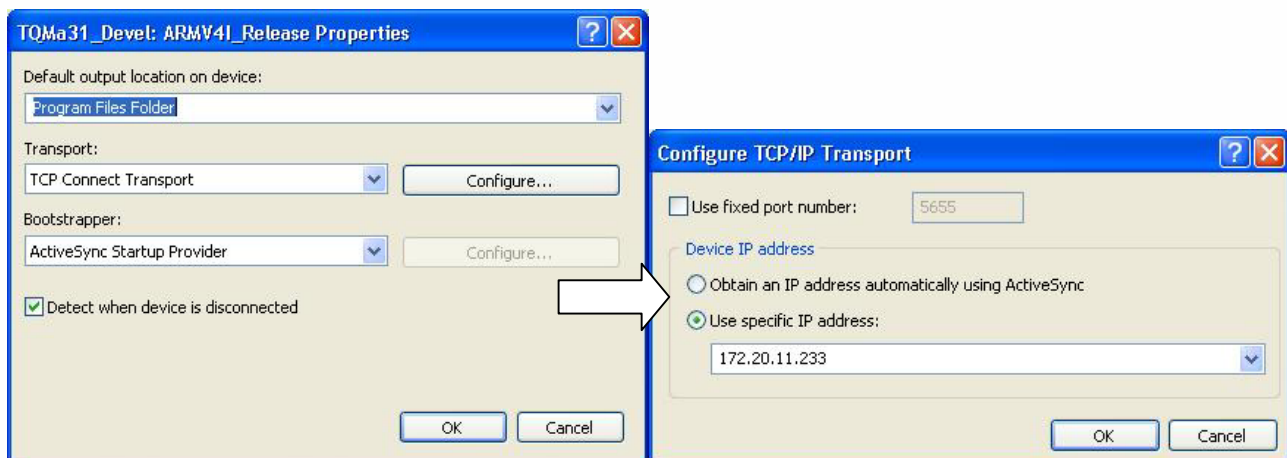


Figure 41: Configure the IP address of the target

Then attach to the device with the button “Connect to Device” on the toolbar.

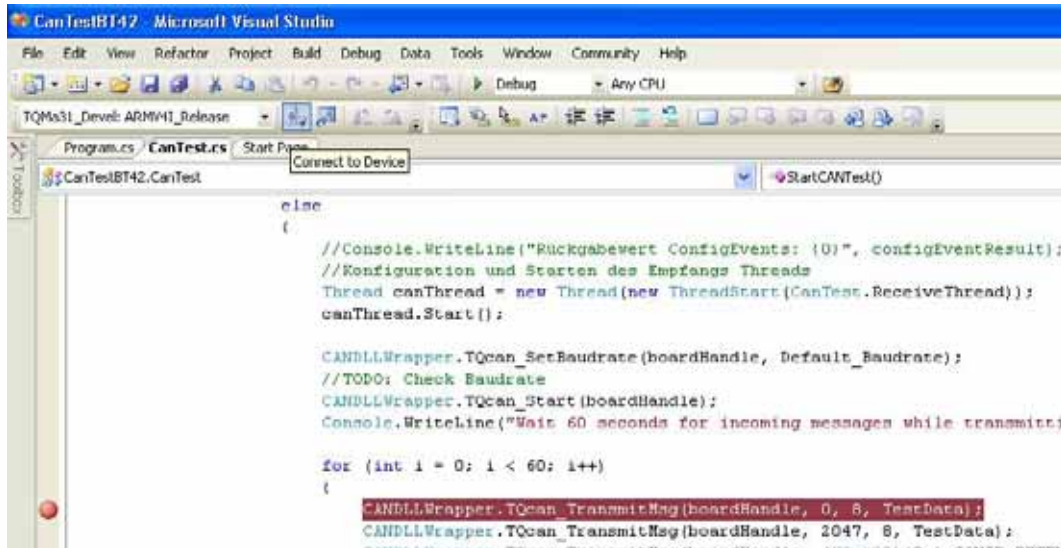


Figure 42: Connect to Device

The “Connecting...” message box appears and reports the connection status.

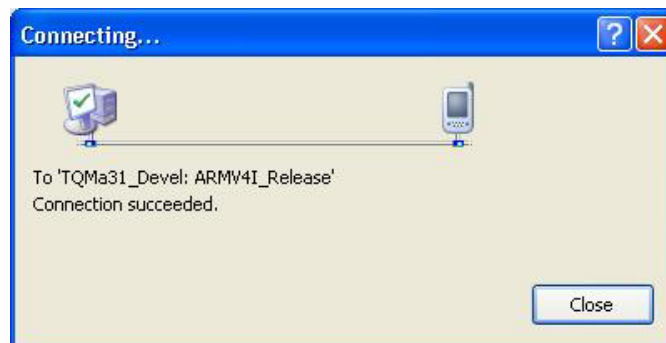


Figure 43: Connection succeeded

Afterwards you can start with debugging by clicking on the button “Start debugging” (green arrow).

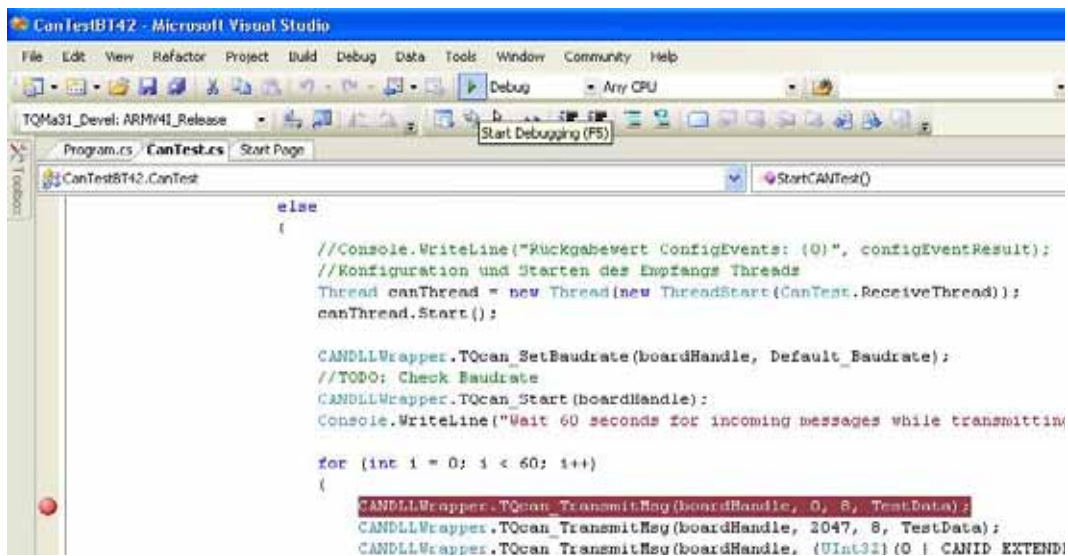


Figure 44: Start Debugging

If you set a breakpoint and the debugger is running, the code execution will be broken. This condition is signaled with a yellow arrow inside the red breakpoint symbol.

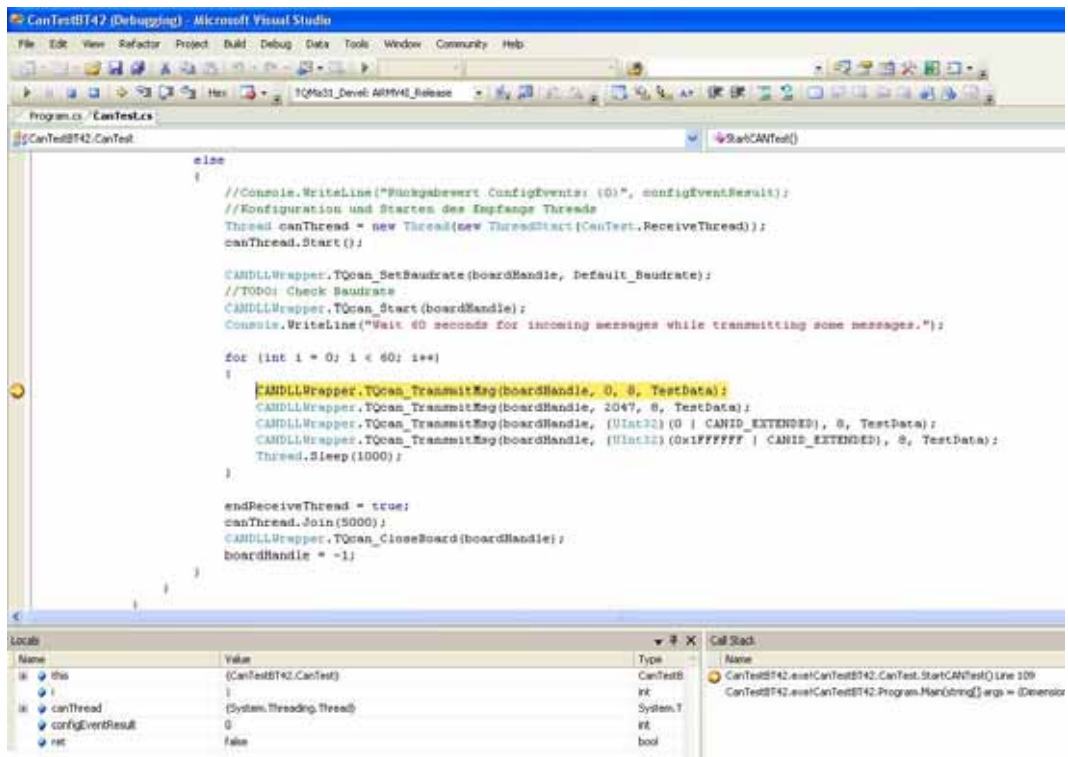


Figure 45: Signaling an active breakpoint

8.4.2.2 Startup and Transport over Ethernet (debugging without Active Sync)

For all systems, that do not have a separate USB client connection, there are 2 possibilities to get the debug connection working without a Active Sync connection. In this case startup and transport runs over the TCP/IP connection.

1. *OpenNETCF Consulting*

offers an executable file. The tool is called "CEDbgSetup.exe" and can be downloaded from <http://community.opennetcf.com/articles/cf/archive/2007/08/31/debugging-without-activesync.aspx>. The tool must be executed on the target device before the "Connect to Device" button from the Visual Studio 2005 toolbar is pressed. Once the tool is running, you can continue with the "Connect to Device" (see chapter Startup over Active Sync / Transport over Ethernet).

2. *Microsoft CMAccept*

also offers a solution, which makes Active Sync needless. You only have to copy the following files to from <Visual Studio 2005 Install Dir>\smartdevices\debugger\target\wce500\<CPU type> to your target:

- clientshutdown.exe
- CMAccept.exe
- ConmanClient2.exe
- DeviceDMA.dll
- eDbgTL.dll
- TcpConnectionA.dll

First run ConmanClient2.exe, afterwards CMAccept.exe. It does not matter if you launch the applications from command line or by double clicking on the icons.

After the tools are running, you can continue with the "Connect to Device".

For further information about the Microsoft solution please see

<http://msdn.microsoft.com/de-de/library/ms228708.aspx>

<http://blogs.msdn.com/vsdteam/archive/2005/01/05/347332.aspx>

8.5 Application deployment

This chapter shows how to include applications and associated files (e.g. DLLs) in your image. Further you will learn how to start your applications automatically when Windows CE has booted.

8.5.1 Adding applications and files to your OS design

You can include your applications and additional files (e.g. DLLs) in your Windows CE image by adding an entry in the MODULES/FILES section of your project.bib file.

E.g. : Include an application named myapp.exe with an additional DLL named myapp.dll and a bitmap file. In this example all files are located in a directory called %_WINCEROOT%\MYAPP\.

Add the following lines to your project.bib file:

```

MODULES
; Name          Path          Memory Type
; -----
MYAPP.EXE      $( _Wincerooot )\MYAPP\MYAPP.EXE  NK  SH
MYAPP.DLL      $( _Wincerooot )\MYAPP\MYAPP.DLL  NK  SH

FILES
; Name          Path          Memory Type
; -----
PLASH.BMP      $( _Wincerooot )\MYAPP\PIC1.BMP  NK  SH

```

MODULES is reserved for executables and DLLs. Any file in the MODULES area will be fixed up to execute in place (XIP). Those files won't be available via the file system. This is recommended especially for DLLs.

All files loaded by applications should be placed in the FILES section. Those files are compressed and available via the file system.

The major difference between the FILES section and the MODULES section is this: If a dynamic-link library (.dll) file is placed in the FILES section, as opposed to the MODULES section, it is loaded into every slot location instead of process slot 1 only, which decreases the virtual address space available to the process.

You can also rename files via the BIB file entries. As you can see in the example above, the bitmap file called PIC1.BMP will show up as SPLASH.BMP in the Windows CE file system.

If you want to learn more about how the Windows CE image is built up and about the different settings for memory types please refer to the Platform Builder help system for "MODULES Section" respectively "FILES Section".

8.5.2 Auto Start-up of Windows CE applications

After the Windows CE boot process has finished, your application must be started. To accomplish this, you have to insert appropriate entries into the *HKEY_LOCAL_MACHINE\init* registry key. This can be done using the *project.reg* file of your workspace.

To create a registry entry for starting an application under the Windows CE, add the following statements to the *HKEY_LOCAL_MACHINE\init* section of your *project.reg* file:

```

[HKEY_LOCAL_MACHINE\init]
"launchXX"="exe_name"
"dependXX"=hex:YY,00

```

Name	Values	Description
launchXX	Name of the application to start.	XX represents a number in decimal from 00 to 99. Only launch numbers from 80 to 89 are valid. All others are reserved.
dependXX	hex:xx,yy	Describes a dependency of the launched application. If

Name	Values	Description
		some other application must be started before the application defined by the launch key can be successfully launched, then this entry should be used to identify that other application. XX,YY represents the launch number in hexadecimal of the application that must be loaded <i>first</i>. (YY is the most significant byte and therefore always 00). One or more dependent applications can be specified per <i>dependXX</i> value. The <i>dependXX</i> entry is optional; if there is no dependency you don't need to use it.

Table 8-1 Application auto start registry settings

The following code example shows a typical Init registry entry using dependencies:

```
[HKEY_LOCAL_MACHINE\Init]
"Launch10"="shell.exe"
"Launch20"="device.exe"
"Launch30"="gwes.exe"
"Depend30"=hex:14,00
"Launch50"="taskman.exe"
"Depend50"=hex:14,00, 1e,00
```

Note:

The application that must be started, first must call the function **SignalStarted()** to inform Windows CE when it is ready. The dependent application cannot run until after the function on which it depends on has issued the **SignalStarted** function.

Note:

If your application is located on an external media (CFCard, USB stick) you have to add the path to this media to the system path variable. Example:

```
[HKEY_LOCAL_MACHINE\init]
"Launch80"="test.exe"
"Depend80"=hex:14,00, 1e,00, 32,00

[HKEY_LOCAL_MACHINE\Loader]
"SystemPath"=multi_sz:"\\Release\\", "\\cfcard\\test\\"
```

8.6 ActiveSync

8.6.1 Required Components

The following components of the Catalog must be added to your OS design:

1. Core OS / Windows CE devices / Applications – End User / File Sync

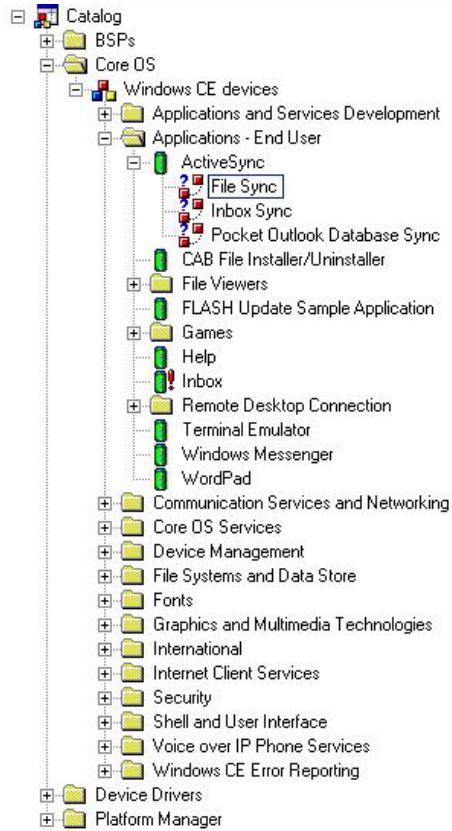


Figure 46: Adding File Sync to an OS design

2. Core OS / Windows CE devices / Shell and User Interface / Network User Interface

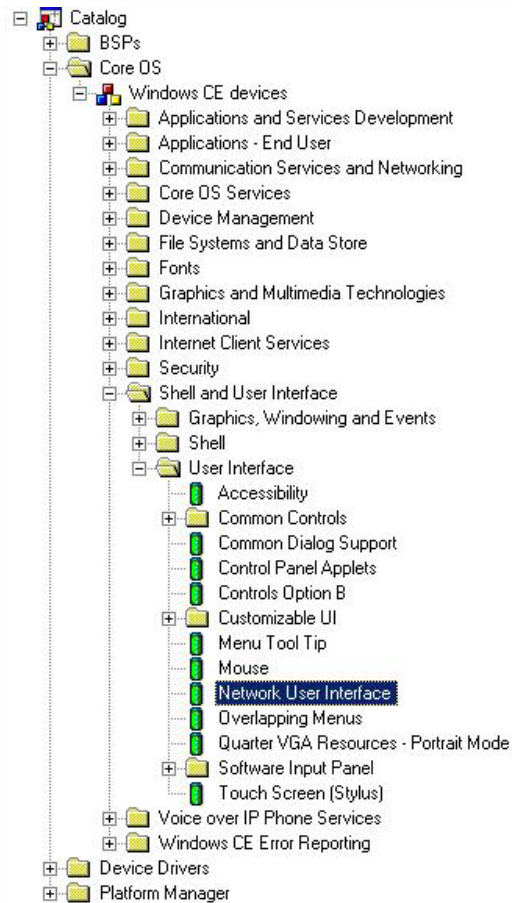


Figure 47: Adding Network User Interface to an OS design

After adding the missing components, build the OS design, create a run-time image and download it to your device.

8.6.2 Establish a Active Sync connection over serial port

On the target device open the start menu, select “Settings” and then “Network and Dial-up Connections”.

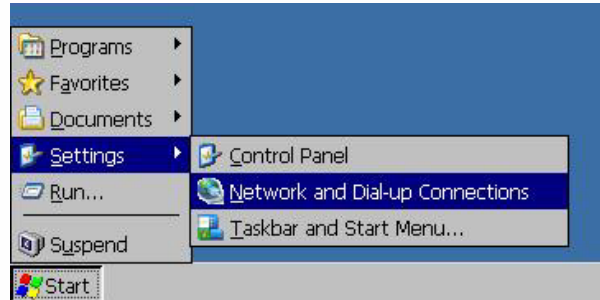


Figure 48: Network and Dial-up Connections

A new window opens with an icon named “Make New Connection”. Make a double click on this icon, in the new dialog type a name for the connection , select type “Direct Connection” and hit “Next >”.

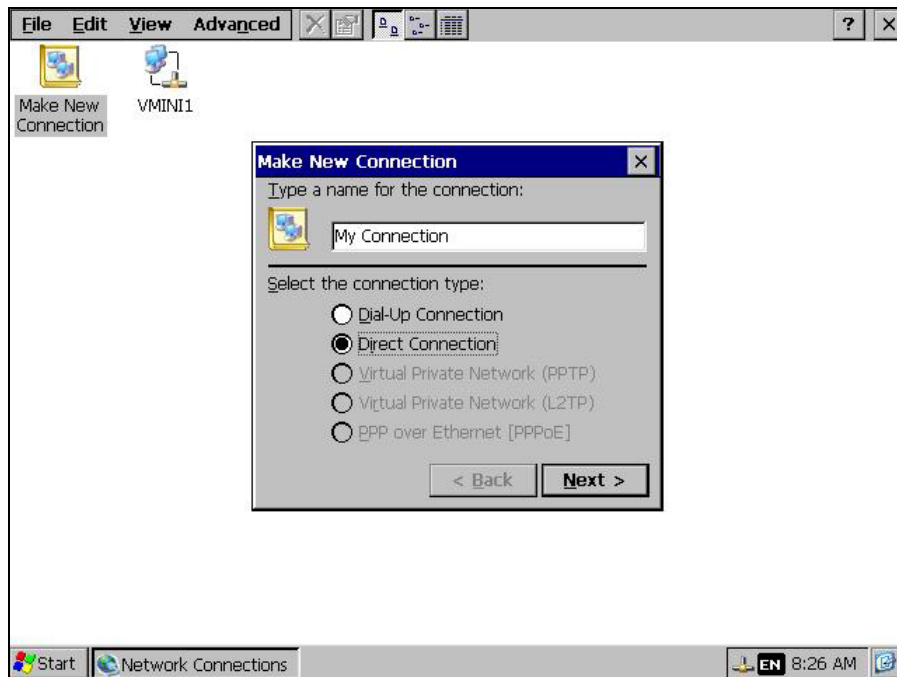


Figure 49: Make a new connection

The next dialog prompts for selecting a device.

After you hit “Finish” a new connection symbol with the name you chose appears in the “Network Connection” window.

Open the control panel and double click on PC connection. Click on the change button and select the connection you have just created.

By double clicking this symbol the system tries to establish a connection to the host PC. If no host PC running ActiveSync is available the device tries to connect a few times, then it waits for incoming connection requests. You can check this by running a terminal program (e.g. Hyperterminal which is included in Windows XP) instead of ActiveSync and set the baud rate to 19200. After clicking the connection symbol on the target device the terminal program should show some "CLIENT" messages it received. Closing the terminal program and running Active Sync will then lead to an established connection.

8.6.3 Establish a ActiveSync connection over USB

Include either "OTG Pure Client Function" or "OTG Full OTG Function" in your OS Design.

Include also the "Serial" USB Function Driver in your OSDesign (**Catalog -> Device Drivers -> USB Function -> USB Function ClientsI**).

When you plug in a Type B Plug in your device and connect the cable with your host computer running ActiveSync the device will be recognized automatically.

9 Additional customizations

The following chapters will give you hints for the usage of some Windows CE components and other useful settings regarding the appearance of Windows CE.

9.1 Windows CE Components

9.1.1 Telnet Server

Telnet allows easy access to your Windows CE device over the network.

To add the Telnet server to your image right-click on the „Telnet Server“ component in the Catalog (**Catalog -> Core OS -> Windows CE devices -> Communication Services and Networking -> Servers**) and choose „Add to OS Design“.

To enable the Telnet server you have to add additional registry keys (e.g. In your project.reg file) as shown below:

```
[HKEY_LOCAL_MACHINE\COMM\TELNETD]
    "IsEnabled"=dword:1          ; set to 0 to disable the telnet server
    "UseAuthentication"=dword:0 ;anonymous access allowed
```

Note:

The settings above will allow anonymous access to your system. There is no password required as long as the *UseAuthentication* Key is set to 0. If you wish to protect the integrity of your system by adding a degree of security, however, you may enable the password feature in Telnet. To do this, set the *UseAuthentication* key to one. The Telnet server will then ask for a login and password before establishing a connection.

To connect via Telnet simply open a Command Prompt in Windows XP and do a Telnet to the IP-address of your target system (e.g. `telnet 192.168.100.25`). You can close the connection by simply typing `exit` at the command prompt.

```

C:\WINDOWS\system32\cmd.exe
Welcome to the Windows CE Telnet Service on WindowsCE
Pocket CMD v 5.0
\> dir

    Directory of \

01/01/98  04:00a  <DIR>      Network
01/01/98  04:00a  <DIR>      SRAMDisk
01/01/98  04:00a  <DIR>      Hard Disk
09/06/06  03:11a  <DIR>      Application Data
09/05/06  07:11p  <DIR>      23 Control Panel.lnk
09/05/06  07:11p  <DIR>      My Documents
09/05/06  07:11p  <DIR>      Program Files
09/05/06  07:10p  <DIR>      Temp
09/05/06  07:10p  <DIR>      Windows

Found 9 file(s). Total size 23 bytes.
1 Dir(s) 122151736 bytes free

\> exit

Verbindung zu Host verloren.
C:\>
  
```

Figure 50: Target connection via Telnet

9.1.2 FTP Server

FTP allows you to transfer files to and from your target system via the File Transfer Protocol without the need of using ActiveSync.

To add the FTP server to your image right-click on the „FTPServer“ component in the Catalog (**Catalog -> Core OS -> Windows CE devices -> Communication Services and Networking -> Servers**) and choose „Add to OS Design“.

To enable the FTP server you have to add additional registry keys (e.g. In your project.reg file) as shown below:

```

[HKEY_LOCAL_MACHINE\COMM\FTPD]
    "IsEnabled"=dword:1
    "UseAuthentication"=dword:1
    "AllowAnonymous"=dword:1
    "AllowAnonymousUpload"=dword:1
    "DefaultDir"="\\"
  
```

Note:

The registry settings shown above allow anonymous access to your target system. This should be disabled before you deploy your OS design. You should set the `AllowAnonymous` registry key to 0 and add a list of allowed users to the registry. Search the Platform Builder Help System for “FTP server UserList” for more information.

To connect via FTP simply open a Command Prompt in Windows XP and do an ftp command to the IP-address of your target system (e.g. `ftp 192.168.100.25`). You can close the connection by simply typing `bye` at the command prompt.

```

C:\>ftp 192.168.100.25
Verbindung mit 192.168.100.25 wurde hergestellt.
220 Service ready for new user.
Benutzer (192.168.100.25:(none)): anonymous
331 Anonymous access allowed, send identity (e-mail name) as password.
Kennwort:
230 User logged in, proceed.
ftp> dir
200 Command okay.
150 File status okay; about to open data connection.
01-01-98 04:00 <DIR> Network
01-01-98 04:00 <DIR> SRAMDisk
01-01-98 04:00 <DIR> Hard Disk
09-06-06 04:43 <DIR> Application Data
09-05-06 20:43 23 Control Panel.lnk
09-05-06 20:43 <DIR> My Documents
09-05-06 20:43 <DIR> Program Files
09-05-06 20:42 <DIR> Temp
09-05-06 20:42 <DIR> Windows
226 Closing data connection.
FTP: 64d Bytes empfangen in 0,36Sekunden 1,24KB/s
ftp> bye
221 Service closing control connection.
C:\>
  
```

Figure 51: Target connection via FTP

9.1.3 Hive Based Registry

If you want to use the registry as a persistent store for your application settings right-click on the „Hive based Registry“ component in the Catalog (**Catalog -> Core OS -> Windows CE devices -> File**

Systems and Data Store -> Registry Storage) and choose „Add to OS

Design“.

Also add the „RAM and ROM File System“(Catalog -> Core OS -> Windows CE devices -> File Systems and Data Store -> File System - Internal).

You will need to set some additional registry keys. These registry settings will tell Windows CE where to place the registry in the File System. The appropriate registry Keys for a Hive on CFCard or SDCard are already prepared.

To select CFCard as hive target device, please set the BSP_HIVE_CFCARD variable in the TQMa31.BAT File (C:\WINCE500\PLATFORM\TQMa31\TQMa31.bat).

To select SDCard as hive target device, please set the BSP_HIVE_SDCARD variable in the TQMa31.BAT File (C:\WINCE500\PLATFORM\TQMa31\TQMa31.bat).

Alternatively you can add the environment variable in the *Environment* tab of the *Platform Settings* dialog.

Note:

Only one of the BSP_HIVE_XXX variables must be set at a time.

Note:

Always perform a clean Build after you have added the Hive-based Registry feature. Choose the *Clean Before Building* option in the *Build OS* menu of the Platform Builder IDE and perform a Sysgen (*Build OS -> Sysgen*).

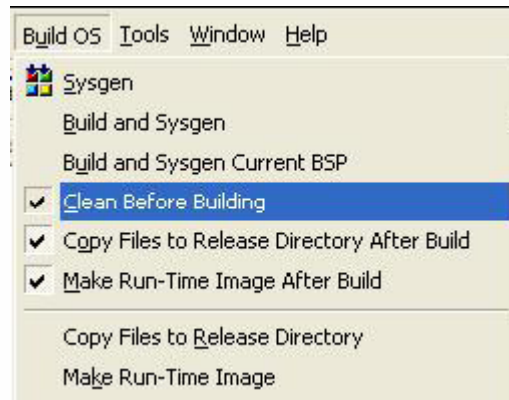


Figure 52: Hive-based Registry - Clean before Building

9.2 Changing des appearance of Windows CE

9.2.1 Changing the desktop wallpaper

1. Add the following key to project.reg:

```
; ***** Wallpaper *****
[HKEY_CURRENT_USER\ControlPanel\Desktop]
"wallpaper"="\\Windows\\customer.bmp"
```

2. Add the file name and path for the new wallpaper bitmap to project.bib:

```
windowsce_qvgap.bmp $( _FLATRELEASEDIR )\customer.bmp NK S
```

3. Copy the bitmap file to the directory
\$(_WINCEROOT)\PBWorkspaces\\$(_TGTPROJ)\WINCE500\TQMa31_ARMV4\OAK\files.

9.2.2 Hiding the taskbar

Add the following keys to project.reg:

```
; ***** Takbar & desktop setting *****
[HKEY_LOCAL_MACHINE\Software\Microsoft\Shell\AutoHide]
; Taskbar auto-hide
@="1"
[HKEY_LOCAL_MACHINE\Software\Microsoft\Shell\OnTop]
; @="1" taskbar always visible (on top)
; @="" taskbar can be covered by other windows
@=""
```

9.2.3 Changing the folder name of storage devices

Example for modifying project.reg:

```
; ***** USB stick folder name *****
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\USBHDPProfile]
    "Folder"="USB Stick"

; ***** CFCard folder name *****
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\PCMCIA]
    "Folder"="CFCard"

; ***** SDCard folder name *****
[HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\SDMemory]
    "Folder"="SDCard"
```

9.2.4 Setting up the time zone to GMT

Add the following registry entries to project.reg:

```
; ***** Date/Time Settings *****
; Time zone default settings
[HKEY_LOCAL_MACHINE\Time Zones]
    "Default"="GMT Standard Time"

; Clock Format
[HKEY_LOCAL_MACHINE\Nls\Overrides]
    "STime"=":"
    "STFmt"="H:mm"

; STime STRING ":" - Time Separator
; STFmt STRING "h:mm:ss tt" - Time Format String
; S1159 STRING "am"
; S2359 STRING "pm"
; Time Format String
; h - Hours 12 Hour Clock with No Leading Zero
; hh - Hours 12 Hour Clock with Leading Zero
; H - Hours 24 Hour Clock with No Leading Zero
; HH - Hours 24 Hour Clock with Leading Zero
; m - Minutes with No Leading Zero
; mm - Minutes with Leading Zero
; s - Seconds with No Leading Zero
; ss - Seconds with Leading Zero
; t - am/pm indicator - First Letter Only
; tt - am/pm indicator - First Two Letters
; ttt - am/pm indicator - First Three Letters
```

10 Appendix

10.1 User interface libraries

10.1.1 CAN (TQ_CanCe.dll)

10.1.1.1 Introduction

TQ_CAN is a Windows CE DLL which provides support functions for the CAN interface of the TQMa31 CPU module.

It provides support for the following functions:

- 1 Functions to open and close the interface (TQcan_OpenBoard/TQcan_CloseBoard)
- 2 Baudrate configuration functions (TQcan_SetBaud)
- 3 Functions to start and stop bus activities (TQCan_Start/TQcan_Stop)
- 4 Functions for reception of bus events and messages (TQcan_EventCount/TQcan_ReadEvent)
- 5 Functions to transmit messages (TQcan_TransmitMsg/TQcan_RequestMsg)

10.1.1.2 Function description

10.1.1.2.1 Return values

All functions return one of the following values (for a detailed description see description of function):

TQc_OK	0	ok
TQc_PARAMETER	-1	illegal parameter passed
TQc_LOADDRIVER	-3	Hardware driver cannot be loaded
TQc_OVERRUN	-2	Transmit Queue full
TQc_LOADDRIVER	-3	Hardware driver cannot be loaded
TQc_DRIVERERROR	-4	Error in driver function
TQc_BUSY	-5	Interface is in use

Table 10-1

10.1.1.2.2 Interface access functions

The functions TQcan_OpenBoard and TQcan_CloseBoard open respectively close the CAN interface of the TQMa31 module.

10.1.1.2.2.1 *TQcan_OpenBoard*

Function: int **TQcan_OpenBoard**(UINT16 unit_no, UINT16 mode);

Description: Initializes the interface with number ,unit_no' (normally: 0) and sets the access mode..

Parameter: unit_no (in)
Number of interface (0..n). At the moment there is only the interface with number 0 available.

mode (in)
Access mode: 0 = normal, TQC_CONSUMER, or TQC_LOM

Return value: >=0 Interface handle
<0 TQc-ReturnCodes (TQc_BUSY, if interface is in use)

Applications can open the interface for reading only by using the mode **TQC_CONSUMER**. This mode does not allow setting of baudrate or message transmission. Useful for bus activity logging.

In **TQC_LOM** mode the CAN interface is switched to "listen-only-mode" (s. MCP2515 documentation). This prevents bus disruption even if baudrate is set wrong.

The return value has to be provided as parameter „board_hdl“ for all other function calls.

10.1.1.2.2.2 *TQcan_CloseBoard*

Function: int **TQcan_CloseBoard**(UINT16 board_hdl);

Description: Release a interface which was opened by a call to TQcan_OpenBoard.

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

Return value: TQc-ReturnCodes (TQc_PARAMETER, , if interface was not opened)

10.1.1.2.3 Bus configuration functions

TQcan_SetBaudrate configures the bus timing.

10.1.1.2.4 TQcan_SetBaudrate

Function: int **TQcan_SetBaudrate**(UINT16 board_hdl,UINT32 Baud);
Description: Set the bus baudrate in bps (Bits per second)
Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

Baud (in)
Wert für die Baudrate
Return value: TQc-ReturnCodes (TQc_PARAMETER, , if interface is not opened)

10.1.1.2.4.1 *TQcan_GetBaudrate*

Function: int **TQcan_GetBaudrate**(UINT16 board_hdl,UINT32 *Baud);
Description: Abfrage der aktuellen Werte der Baudrate in bps (Bits per second)
Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

Baud (out)
Zeiger auf die aktuelle Baudrate
Return value: TQc-ReturnCodes (TQc_PARAMETER, , if interface is not opened)

10.1.1.2.5 Message acceptance filter

The functions `TQcan_SetAccMask` and `TQcan_SetAccCode` allow programming of the message acceptance filter to lower CPU load through unrequested messages.

Formula for message acceptance:

Accept If $(id \wedge code) \& mask == 0$

The acceptance mask is used to select the bits which are evaluated by the filter (0=don't care). The acceptance code defines the bits of the id which have to be set to be accepted.

The MSB (`CANID_EXTENDED=0x80000000`) decides if the filter is used for extended IDs..

Some examples:

	Standard	Extended
Open	<code>acc_mask=0x000</code> <code>acc_code=0x000</code>	<code>acc_mask=0x80000000</code> <code>acc_code=0x80000000</code>
Close	<code>acc_mask=0x7FF</code> <code>acc_code=0x000</code>	<code>acc_mask=0x9FFFFFFF</code> <code>acc_code=0x80000000</code>
One(*)	<code>acc_mask=0x7FF</code> <code>acc_code=0x001</code>	<code>acc_mask=0x9FFFFFFF</code> <code>acc_code=0x80000001</code>
Odd (*)	<code>acc_mask=0x001</code> <code>acc_code=0x001</code>	<code>acc_mask=0x80000001</code> <code>acc_code=0x80000001</code>
Even (*)	<code>acc_mask=0x001</code> <code>acc_code=0x000</code>	<code>acc_mask=0x80000001</code> <code>acc_code=0x80000000</code>

The filter „Open“ accepts all messages, as `acc_mask=0`; the filter „Close“ accepts no message (but Id 0) As all bits are used for filtering

The filter „One“ accepts only messages with ID 1.

The filter „Odd“ accepts only odd IDs, the filter „Even“ only even IDs.

10.1.1.2.5.1 `TQcan_SetAccMask`

Function: `int TQcan_SetAccMask(UINT16 board_hdl,UINT32 acc_mask);`

Description: Set the acceptance mask register.

Parameter: `board_hdl` (in)
Interface handle (s. `TQcan_OpenBoard`)

`acc_mask` (in)
Value of the acceptance mask register to set; MSB (`0x80000000`) is set for extended message IDs.

Return value: `TQc-ReturnCodes` (`TQc_PARAMETER`, if interface is not opened)

10.1.1.2.5.2 *TQcan_SetAccCode*

Function: int **TQcan_SetAccCode**(UINT16 board_hdl,UINT32 acc_code);

Description: Set the acceptance code register.

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

acc_code (in)
Value of the acceptance code register to set; MSB (0x80000000) is set for extended message IDs.

Return value: TQc-ReturnCodes (TQc_PARAMETER, if interface is not opened)

10.1.1.2.5.3 *TQcan_GetAccMask*

Function: int **TQcan_GetAccMask**(UINT16 board_hdl,UINT32 *acc_mask);

Description: Query the acceptance mask register.

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

acc_mask (out)
Current value of the acceptance mask register; MSB (0x80000000) is set for extended message IDs.

Return value: TQc-ReturnCodes (TQc_PARAMETER, if interface is not opened)

10.1.1.2.5.4 *TQcan_GetAccCode*

Function: int **TQcan_GetAccCode**(UINT16 board_hdl,UINT32 *acc_code);

Description: Query the acceptance code register.

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

acc_code (out)
Current value of the acceptance code register; MSB (0x80000000) is set for extended message IDs.

Return value: TQc-ReturnCodes (TQc_PARAMETER, if interface is not opened)

10.1.1.2.6 Control of bus activity

The functions TQcan_Start/TQcan_Stop start respectively stop the bus activity. The function TQcan_CanStatus queries the current bus state.

10.1.1.2.6.1 TQcan_Start

Function: int **TQcan_Start**(UINT16 board_hdl);

Description: Starts bus communication with current settings (baudrate). Message transmission/reception is possible after call of this function.

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

Return value: TQc-ReturnCodes (TQc_PARAMETER, if interface is not opened)

10.1.1.2.6.2 TQcan_Stop

Function: int **TQcan_Stop**(UINT16 board_hdl);

Description: Stops bus communication. Current settings persist. Message transmission/reception is stopped.

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

Return value: TQc-ReturnCodes (TQc_PARAMETER, if interface is not opened)

10.1.1.2.6.3 TQcan_CanStatus

Function: int **TQcan_CanStatus**(UINT16 board_hdl);

Description: Returns the current bus state encoded (s. STS_xx constants below)

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

Return value: >=0 bus state (bitmask)
<0 TQc-ReturnCodes (TQc_PARAMETER, if interface is not opened)

STS_RUNNING	0x01
STS_TX_PENDING	0x08
STS_CAN_INIT	0x10
STS_ERRLVL	0x20
STS_ERRLVL	0x40
STS_BUSOFF	0x80

Table 10-2

10.1.1.2.7 Message transmission

The functions `TQcan_TransmitMsg` and `TQcan_RequestMsg` allow the transmission of CAN messages (respectively remote request messages). `TQcan_Pending` queries the number of messages still waiting for transmission.

10.1.1.2.7.1 *TQcan_TransmitMsg*

Function: int `TQcan_TransmitMsg`(UINT16 board_hdl,UINT32 id,UINT8 len,UINT8 *data);

Description: Send a CAN message with ID „id“ and length „len“.

Parameter: board_hdl (in)
Interface handle (s. `TQcan_OpenBoard`)

id (in)
CAN-Id of the requested message (set MSB for extended IDs)

len (in)
Number of data bytes (0-8)

data (in)
pointer to data to send

Return value: TQc-ReturnCodes (TQc_OVERRUN if transmit buffer is full)

10.1.1.2.7.2 *TQcan_RequestMsg*

Function: int `TQcan_RequestMsg`(UINT16 board_hdl,UINT32 id,UINT8 len);

Description: Send a remote request for the CAN message with ID „id“ and length „len“.

Parameter: board_hdl (in)
Interface handle (s. `TQcan_OpenBoard`)

id (in)
CAN-Id of the requested message (set MSB for extended IDs)

len (in)
Number of data bytes (0-8)

Return value: TQc-ReturnCodes (TQc_OVERRUN if transmit buffer is full)

10.1.1.2.7.3 *TQcan_Pending*

Function: int `TQcan_Pending`(UINT16 board_hdl);

Description: Queries number of messages waiting for transmission in the send buffer.

Parameter: board_hdl (in)
Interface handle (s. `TQcan_OpenBoard`)

Return value: >=0 = Number of queued messages
<0 =TQc-ReturnCodes

10.1.1.2.8 Message reception

All received CAN messages and bus errors are stored in TQc_CAN_EVT structures. The receive event queue is configured via TQcan_ConfigEvent function. TQcan_ReadEvent reads the received events from the receive queue.

10.1.1.2.8.1 The TQc_CAN_EVT struct

All bus events are returned in a TQc_CAN_EVT struct:

```
//-----
// Structure of a CAN event
//-----
typedef struct
{
    BYTE    evt_INTF;           // MCP_INTF register contents
    BYTE    evt_EFLG;           // MCP_EFLG register contents
    __int64 evt_time;           // TimeStamp (in µs)
    ULONG Id;                   // the CanID (plus CANID_EXTENDED|CANID_REMOTE)
    BYTE    Dlc;                // Data Length code
    BYTE    CanNum;             // # of the CanBus (currently always zero)
    BYTE    Data[8];            // The Data
} TQc_CAN_EVT;

//-----
// DEFINITION of the MCP2515 CANINTF register contents
// see TQc_CAN_EVT.evt_INTF and TQcan_ConfigEvents(evt_mask)
//-----
# define MCP_INTF_RX0IF    0x01    // INTF.0 = RX0IF = Receive Buffer 0 Full Interrupt
# define MCP_INTF_RX1IF    0x02    // INTF.1 = RX1IF = Receive Buffer 1 Full Interrupt
# define MCP_INTF_TX0IF    0x04    // INTF.2 = TX0IF = Transmit Buffer 0 Empty Interrupt
# define MCP_INTF_TX1IF    0x08    // INTF.3 = TX1IF = Transmit Buffer 1 Empty Interrupt
# define MCP_INTF_TX2IF    0x10    // INTF.4 = TX2IF = Transmit Buffer 2 Empty Interrupt
# define MCP_INTF_ERRIF    0x20    // INTF.5 = ERRIF = Error Interrupt Flag bit
# define MCP_INTF_WAKIF    0x40    // INTF.6 = WAKIF = Wakeup Interrupt
# define MCP_INTF_MERRF    0x80    // INTF.7 = MERRF = Message Error Interrupt
# define MCP_INTF_ALL      0xFF

//-----
// DEFINITION of the MCP2515 EFLG register contents
// see TQc_CAN_EVT.evt_EFLG
//-----
# define MCP_EFLG_EWARN    0x01    // EFLG.0 = EWARN = Error Warning Linit reached (>96 errors-TX or RX)
# define MCP_EFLG_RXWAR    0x02    // EFLG.1 = RXWAR = Receive Error Warning (>96 erros - RX)
# define MCP_EFLG_TXWAR    0x04    // EFLG.2 = TXWAR = Transmit Error Warning (>96 erros - TX)
# define MCP_EFLG_RXEP     0x08    // EFLG.3 = RXEP = Receive Error-Passive (>128 erros - RX)
# define MCP_EFLG_TXEP     0x10    // EFLG.4 = TXEP = Transmit Error-Passive (>128 erros - TX)
# define MCP_EFLG_TXB0     0x20    // EFLG.5 = TXB0 = Bus-Off Error (>255 erros - TX)
# define MCP_EFLG_RX0OVR    0x40    // EFLG.6 = RX0OVR = Receive Buffer 0 Overflow
# define MCP_EFLG_RX1OVR    0x80    // EFLG.7 = RX1OVR = Receive Buffer 1 Overflow
```

The elements „evt_INTF“ und „evt_EFLG“ reflect the MCP2515 hardware at the time the event occurred..

10.1.1.2.8.2 TQcan_ConfigEvents

Function: int **TQcan_ConfigEvents**(UINT16 board_hdl,UINT16 que_size,UINT8 evt_mask,HANDLE hNotify,UINT16 nLimit);

Description: initialize the receive event queue.

Parameter: board_hdl (in)
Interface handle

que_size (in)
size of event queue (max. messages count).

evt_mask (in)
mask of events to buffer in queue (s. MPC_IR_xx defines).

hNotify (in, opt)
WIN32 event handle to call if more than „nLimit“ event are queued.

nLimit (in, opt)
notofication threshold.

Return value: TQc-ReturnCodes

10.1.1.2.8.3 TQcan_EventCount

Function: int **TQcan_EventCount** (UINT16 board_hdl);

Description: query number of event in receive buffer

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

Return value: >=0 = number of received events
<0 =TQc-ReturnCodes

10.1.1.2.8.4 TQcan_ReadEvent

Function: int **TQcan_ReadEvent** (UINT16 board_hdl, UINT16 count, TQc_CAN_EVT *p_evt));

Description: read one ore more events from receive buffer

Parameter: board_hdl (in)
Interface handle (s. TQcan_OpenBoard)

count (in)
Count of events to read (size of array p_evt)

p_evt (out)
Pointer to an array of „count“ event structures.

Return value: ≥ 0 = Number of received messages.
 < 0 = TQc-ReturnCodes

10.1.1.2.8.5 *TQcan_ResetTimestamp*

Function: `int TQcan_ResetTimestamp(UINT16 board_hdl);`

Description: Resets the time for received messages to 0.

Parameter: `board_hdl` (in)
 Interface handle (s. TQcan_OpenBoard)

Return value: TQc-ReturnCodes (TQc_PARAMETER, if interface is not opened)

10.1.2 General Purpose Input/Output (TQ_gpio.dll)

10.1.2.1 Introduction

TQ_gpio is a Windows CE DLL which provides support functions for the GPIO features of the TQMa31 CPU module.

There are fundamentally next functions groups:

- 1 Open and close the driver (TQgpio_Open / TQgpio_Close)
- 2 Set and get GPIO I/O (TQgpio_SetPortValue / TQgpio_GetPortValue)
- 3 Configure GPIO Inputs as Interrupt (TQgpio_ConfigurePortInterrupt / TQgpio_AcknowledgeInterrupt)

10.1.2.2 Functions description

10.1.2.2.1 Return value

All functions return the next value (detailed information can be find in each function).

TQc_OK	0	ok
TQc_PARAMETER	-1	illegal parameter passed
TQc_LOADDRIVER	-3	Hardware driver cannot be loaded
TQc_DRIVERERROR	-4	Error in driver function

Table 10-3

10.1.2.2.2 Open and close the driver

With the functions TQgpio_Open and TQgpio_Close, the application reserves and releases the GPIO driver.

10.1.2.2.2.1 *TQgpio_Open*

Function : `UINT32 TQgpio_Open();`

Description : Request access to GPIOs

Return value: `>=0` -handle
 `-1` – invalid board handle

The returned handle should be used as parameter for calling the next functions.

10.1.2.2.2.2 *TQgpio_Close*

Function : `UINT32 TQgpio_Close(UINT32 handle);`

Description : A board registered with 'TQgpio_Open' can be deregistered with 'TQgpio_Close'. This includes the reset of the hardware and deinstallation of the possibly used interrupt.

Parameters: handle (in)
 Identification handle.

Return value: TQcan return codes.
 TQc_PARAMETER if invalid params passed

10.1.2.2.3 Set and get GPIO I/O

Before these actions can be taken, a handle to the TQgpio must already be opened. The set and get functions manage the values in the GPIO Inputs and Outputs.

10.1.2.2.3.1 *TQgpio_SetPortValue*

Function : int TQgpio_**SetPortValue**(UINT32 handle, UINT32 output, UINT8 data);

Description : Set a output to the value given as parameter

Parameters: handle (in)
Identification handle.

output (in)
the output (GPO1 - GPO8)

data (in)
value to set (0 or 1)

Return value: TQcan return codes.
TQc_PARAMETER if invalid params passed

10.1.2.2.3.2 *TQgpio_GetPortValue*

Function : int TQgpio_**GetPortValue**(UINT32 handle, UINT32 output, UINT8 *p_data)

Description : Read value from a port (input or output)

Parameters: handle (in)
Identification handle.

output (in)
the output (GPO1 - GPO8 or GPI1 - GPI6)

p_data (out)
value read from port (0 or 1)

Return value: TQcan return codes.
TQc_PARAMETER if invalid params passed

10.1.2.2.4 Configure GPIO Inputs as Interrupt

Before these actions can be taken, a handle to the TQgpio must already be opened.

10.1.2.2.4.1 *TQgpio_ConfigurePortInterrupt*

Function : int TQgpio_**ConfigurePortInterrupt**(UINT32 handle, UINT32 input, UINT32 type, HANDLE hEvent);

Description : Configure port for interrupt generation

Parameters: handle (in)
Identification handle.

input (in)
the input to configure (GPI1 - GPI6)

type (in)
interrupt type (level/edge low-active/high-active)
value: 0 – 4 (see the table below)

hEvent (in)
event handle (this event will be signalled, when the interrupt occurs)

Return value: TQcan return codes.
TQc_PARAMETER if invalid params passed
TQc_DRIVERERROR if interrupt is already configured (use INTERRUPT_NONE to free it first)

INTERRUPT_LEVEL_LOW_ACTIVE	0
INTERRUPT_LEVEL_HIGH_ACTIVE	1
INTERRUPT_RISING_EDGE	2
INTERRUPT_FALLING_EDGE	3
INTERRUPT_NONE	4

Table 10-4: Interrupt types

Before the interrupt is configured, a task should be waiting for the event.

10.1.2.2.4.2 *TQgpio_AcknowledgeInterrupt*

Function : int TQgpio_**AcknowledgeInterrupt**(UINT32 handle, UINT32 input);

Description : Acknowledge interrupt

Parameters: handle (in)
Identification handle.

input (in)
the input to acknowledge (GPI1 - GPI6)

Return value: TQcan return codes.
TQc_PARAMETER if invalid params passed

10.2 Third Party Software

10.2.1 PrinterCE.NETCF SDK for Visual Studio 2005

The PrinterCE.NETCF SDK for Visual Studio 2005 add powerful printing functionality to your apps developed with Microsoft's Visual Studio 2005 .Net CF 2.0 (Compact Framework 2.0) development environment.

Features:

- Works with C#, VB.Net and other .NetCF languages
- Easy to use - forget device contexts, bit-blitting & other complexities
- Supports VS2008, VS2005, VS2003, Compact Framework 3.x, 2.x, 1.x
- Text - Select font, size, style, color, rotation, page position (or let PrinterCE automatically position multiple rows of text for you)
- Auto-word wrap and page feed text printing options
- Drawing objects - lines, ellipses, rectangles, rounded rectangles - Select size, location, line width, color, fill
- Images - BMP & JPG support (plus GIF & PNG on Pocket PCs) - Select size, aspect ratio, rotation
- Full color printing
- Print using Infrared, Bluetooth, serial, and network connections

Note:

Licensing costs for the PrinterCE.NETCF SDK must be paid separately.

10.2.2 Integration

1. Add the file names and appropriate paths to project.bib:

FILES

Name	Path	Memory	Type
PrCEDemo_NetCF.exe	\$(_WINCEROOT)\PBWorkspaces\\$_TGTPROJ\WINCE500\TQMa31_ARMV4I\OAK\files\PrCEDemo_NetCF.exe	NK	U
PrCE_NetCF.dll	\$(_WINCEROOT)\PBWorkspaces\\$_TGTPROJ\WINCE500\TQMa31_ARMV4I\OAK\files\PrCE_NetCF.dll	NK	SH
PrinterCE.NetCF.dll	\$(_WINCEROOT)\PBWorkspaces\\$_TGTPROJ\WINCE500\TQMa31_ARMV4I\OAK\files\PrinterCE.NetCF.dll	NK	SH

2. Copy the driver files PrCE_NetCF.dll, PrinterCE.NetCF.dl and the demo application PrCEDemo_NetCF.exe to the directory
\$_WINCEROOT)\PBWorkspaces\\$_TGTPROJ\WINCE500\TQMa31_ARMV4I\OAK\files.

Note:

For additional information about integration of printer support please see IP network printer and USB printer support for additional catalog components.

10.2.3 Functional range

Please see the PrinterCE.NetCF Developer's Guide
http://www.fieldsoftware.com/PrinterCE_NetCF_Docs.htm

10.2.4 Supported Printers

Please see <http://www.fieldsoftware.com/PrintersSupported.htm>